The Journal of Systems & Software 177 (2021) 110944

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Adapting Behavior Driven Development (BDD) for large-scale software systems ☆

Mohsin Irshad [a,b,*], Ricardo Britto [a,b], Kai Petersen [b]

[a] Ericsson AB, Sweden
[b] Blekinge Institute of Technology, Sweden

ARTICLE INFO

Article history:
Received 14 May 2020
Received in revised form 28 February 2021
Accepted 5 March 2021
Available online 13 March 2021

Keywords:
Behavior-driven
Large-scale
BDD
Software processes
System of systems

ABSTRACT

Context: Large-scale software projects require interaction between many stakeholders. Behavior-driven development (BDD) facilitates collaboration between stakeholders, and an adapted BDD process can help improve cooperation in a large-scale project.
Objective: The objective of this study is to propose and empirically evaluate a BDD based process adapted for large-scale projects.
Method: A technology transfer model was used to propose a BDD based process for large-scale projects. We conducted six workshop sessions to understand the challenges and benefits of BDD. Later, an industrial evaluation was performed for the process with the help of practitioners.
Results: From our investigations, understanding of a business aspect of requirements, their improved quality, a guide to system-level use-cases, reuse of artifacts, and help for test organization are found as benefits of BDD. Practitioners identified the following challenges: specification and ownership of behaviors, adoption of new tools, the software projects' scale, and versioning of behaviors. We proposed a process to address these challenges and evaluated the process with the help of practitioners.
Conclusion: The evaluation proved that BDD could be adapted and used to facilitate interaction in large-scale software projects in the software industry. The feedback from the practitioners helped in improving the proposed process.

1. Introduction

Software development is a complex process that involves various stakeholders and their interaction to conceptualize, plan, develop, test, and release a software product (Pressman, 2005). These days, software products transform existing businesses into more agile, customer-oriented, and robust business setups by automating manual processes (Earley, 2014). These products are developed to support a new or existing commercial business (e.g., industrial automation, telecommunication, robotics).

Large-scale software products are a fundamental part of this transformation of existing businesses, and products are becoming software intensive with time. These large-scale products are developed in large-scale projects (i.e., Dikert et al. (2016) defined it as *software development projects with 50 or more people or at least six teams.* Dikert et al. (2016). However, developing these large-scale software systems has proven to be a challenging task because of their characteristics such as parallel development (Perry et al., 2001), distributed teams (Bass, 2015), misunderstood requirements (Kasauli et al., 2017), and effort estimation issues (Usman et al., 2018). Researchers have explored possible solutions to improve the development of large-scale products, such as by reducing large-scale systems into smaller manageable systems (Bui-Thanh et al., 2008), by improving the requirements engineering process (Konrad and Gall, 2008), or by using knowledge engineering (Scacchi, 1989) to facilitate the development process. Large-scale system testing is a complex process that requires interaction, communication, and verification activities spreading over various sub-products (De Almeida et al., 2010). As there are multiple sub-products involved in testing large-scale systems, the synchronization of test activities and release management are challenges associated with testing large-scale systems (Linares-Vásquez et al., 2017). In the context of this study, a sub-product is an independent unit of deployment (Obbink et al., 2002), e.g., a micro-service, a component, or a library.

Dikert et al. (2016) described that in large-scale projects, requirements are defined at a very abstract level, and it may be a problem to transform the high-level requirements to correct functional requirements. The alignment of requirements and verification phase is necessary to succeed in software development,

☆ Editor: Antonia Bertolino.

* Correspondence to: Software Engineering Research Lab, Blekinge Institute of Technology, Valhallavägen 1, 371 41 Karlskrona, Sweden.
E-mail addresses: mohsin.irshad@bth.se (M. Irshad), ricardo.britto@bth.se (R. Britto), kai.petersen@bth.se (K. Petersen).

https://doi.org/10.1016/j.jss.2021.110944

as described by Sabaliauskaite et al. (2010). Behavior-driven development (referred to as BDD in this study) is a test-driven development methodology to align the business and technical needs of software (North, 2006a). In traditional software development organizations, the business needs are unknown to the developers, testers, and other engineers working on these products (Sabaliauskaite et al., 2010). There is a gap between the technical and business aspects of software development.

Initially, when BDD was introduced, the objective was to provide a systematic process that supports a common vocabulary and shared understanding of requirements among customers, requirements engineers, developers, and testers (North, 2006a). The use of test cases as specifications placed this methodology into the category of acceptance test-driven development methodologies (e.g., story-driven development, specification driven development) (Lazar et al., 2010). Acceptance test-driven development is commonly used to bridge the gap between the business of the customer and technical aspects of software (Melnik and Maurer, 2007; Otaduy and Díaz, 2017).

BDD was aimed at normal-scale software projects, without multiple teams or sites (North, 2006a). Furthermore, the current state of the art and practice lacks solutions or empirical evidence related to BDD in large-scale software projects. The majority of the studies propose theoretical concepts associated with BDD that are not yet evaluated in the industry and cover only a small part of a software development cycle (e.g., implementation and testing phases). While BDD is an approach that might improve software development in large-scale contexts, the empirical evidence of BDD's usefulness in large-scale projects is missing.

As BDD aims to enhance communication between different stakeholders, it is crucial to propose a BDD-based process adapted for large-scale software development. In this study, we have introduced a BDD based process for large-scale software development and performed an empirical investigation of the benefits and challenges associated with the introduction of BDD in large-scale software projects. We start our investigation by understanding the existing development process, identifying the challenges and benefits of BDD with the help of several workshop sessions. Based on the outcome of workshop sessions, we propose and evaluate a BDD based process using industrial practitioners. They (practitioners) perceived the process as useful for large-scale software development and suggested improvements in the process. Later, based on the feedback of the practitioners, we revised the proposed software development process.

The study provides background and related work on behavior-driven development in large-scale development in Section 2. The research design and analysis are present in Section 3. Section 4 describes the results and the discussion related to the results is present in Section 5. The threats to the validity of our investigation are presented in Section 6, and Section 7 contains conclusions and view on future work.

2. Background and related work

This section briefly describes the main concepts used in this study, such as large-scale software development and behavior-driven development. Furthermore, in the related work section, we have described the studies that have used BDD practices in a software development process.

2.1. Definition of behavior driven development

In BDD, requirements are described in two parts, the central part referred to as 'Narrative or feature or behavior' and the acceptance criteria referred to as "Scenario" (North, 2006b). There can be multiple scenarios under one behavior/feature. An example using a BDD template suggested by North (2006b) is shown below:

Behavior/Feature:
As a user
I want to login
So that I can view products
Scenario 1: A user is able to login to the system
Given User enters its username and password
When authentication is successful
Then user logs-in to the system
Scenario 2: A user is able to view products
Given User authenticates with correct credentials
When user logs-in to the system
Then user can view the products

These behaviors (also known as features) are reported in feature files, and these feature files act as test and requirement documents (Gohil et al., 2011). The vocabulary in the feature file is used by the stakeholders (e.g., product managers, developers, testers, architects) when communicating about the behaviors. In this way, feature files promote the common understanding of the requirements (Gohil et al., 2011).

Jørgensen (2018) suggested that agile software development practices (XP, Scrum, Test-Driven Development (TDD), etc.) are more suited for large-scale software development. However, these practices introduce new challenges that force the modification of existing practices to form new practices. Behavior-driven development is one such example, which attempts to improve the TDD by adding collaboration and coordination among all the stakeholders. Previous research has shown that the communication gap leads to failure to meet customer requirements (Bjarnason et al., 2011). Behavior-driven development's main features are user-acceptance criteria provided by the customer and to foster a common vocabulary in the organization to improve knowledge sharing, coordination, and collaboration. These features make BDD a good fit for large-scale software development organizations that want to enhance the customer's perspective and knowledge sharing in the organization.

Solis and Wang identified six characteristics of BDD (Solis and Wang, 2011), and these characteristics are:

- the use of ubiquitous language based on the business terminologies,
- iterative decomposition process for the high-level specifications,
- templates to write user stories and scenarios,
- automated acceptance tests,
- readable specification code,
- elaboration of behaviors based on the needs of the development phase

In their literature review, Solis and Wang found that a limited number of studies are present on BDD, and considerable work is needed to evaluate the BDD (Solis and Wang, 2011) in different contexts. BDD's characteristics are still under investigation by the research community, and a limited number of studies have described the use of behavior-driven development in an industrial context.

The Test-Driven development inspires BDD principles, and behaviors (or features) are specified at the start of the development. These behaviors fail at the beginning since no product development is done yet. As the product development starts and the functionality is implemented in the product, the failing behaviors start to pass (North, 2006a). Here it is important to note that these behaviors (also known as"features") are specified

based on the business needs of the customer (North, 2006b). BDD does not specify an exact format for behaviors, although behaviors are often described using domain-specific languages (e.g., Gherkin) (Egbreghts, 2017).

2.2. Large-scale software development

Large-scale software development is defined as a development process in which several teams are involved. These teams can be located across different sites and collaborate to develop a joint product or system of systems, as defined by Dingsøyr et al. (2013) and by Dikert et al. (2016). The dependency of customer requirements on various sub-products of a large-scale software product makes the development and verification process difficult to manage. The difficulties in managing, developing, and delivering large-scale projects are extensively described in the research literature on large-scale product development (Dikert et al., 2016). Helgesson et al. (2019) conducted an empirical study that identifies cognitive load drivers (i.e., tools, information, and work processes) as an issue for software practitioners in a large-scale software project. Britto et al. (2019) conducted an empirical study on the evaluation of developers in large-scale projects. Requirements engineering-related challenges are the most mention challenges in the research literature on large-scale development (Dikert et al., 2016) because of extensive co-ordination that is required to write and communicate the software requirements correctly.

The software industry is using various solutions to address problems related to large-scale software development. Smite et al. (2019) described a case-study from a large software organization that has used “Guilds” as a means to improve communication and coordination in an organization. In an industrial study on large-scale software development projects, Ali et al. (2016) addressed the communication and information related challenges by using the value stream mapping. Dingsøyr et al. (2017) found that a combination of scheduled and unscheduled meetings helps improve communication and coordination in large-scale software development projects. Kettunen and Laanti (2008) suggested that to succeed in large-scale product development, a holistic view covering all the phases of software development is needed.

Recent research in large-scale software development has focused on the individual parts of software development processes, and very few studies address the complete software development life cycle. Coordination among different phases such as requirements, development, and verification phase is crucial for successfully delivering large-scale software development projects. The results from a survey by Begel et al. (2009) show that 98% of software practitioners need to coordinate closely with other teams to deliver a successful large-scale software product. Usman et al. (2018) found that in large-scale software projects, effort estimation often overruns because of the difficulties related to coordination in multi-site development settings.

Stray et al. (2019) provides empirical evidence that in large-scale software development, meetings such as Scrum of Scrums are not enough for coordination between different teams. They suggested that additional coordination practices are required for the successful delivery of large-scale software development projects. Customer collaboration and knowledge sharing are two of the eight research areas in large-scale software development that Dingsøyr and Moe (2013) suggested for future research. In this study, we have attempted to propose and evaluate these additional coordination practices, with BDD’s help, in a large-scale software development context.

2.3. Large-scale software testing

Large-scale software testing leads to several challenges. Ali et al. (2012) investigated the practices and challenges of testing a complex system of systems in the telecommunication domain. They observed that tests take place on multiple test levels, where defect slippage occurs between the levels. That is, a defect that should have been found in one level (e.g., unit testing) is located at a later level (e.g., system test). Challenges in three areas (fault slippage, maintenance of tests, and timeliness of testing) were observed. Fault slippage was caused by the specific challenge in large-scale development, such as unclear division of responsibilities between test levels, vague requirements, and poor quality tests. Vierhauser et al. (2014) also investigated the challenges and practices in the large-scale system of systems testing. They found that practices are team specific and that the tools used in development were project-dependent. System size and complexity as an issue concerning maintenance were also highlighted. Furthermore, communication effort, difficulty in understanding systems specifications, issues concerning tools, and coordination were the challenges that were specified. Overall, the findings of both studies (Ali et al., 2012) and (Vierhauser et al., 2014) are well aligned. Minhas et al. (2020) investigated practices and challenges with a focus on regression testing, highlighting time to test, the maintenance of the test suit, communication, and issues in test selection and prioritization as challenges.

Various solutions have been proposed to test large-scale software systems. Concerning testing measurement and estimation, Obara et al. (1996) highlighted the importance of measuring the process, including the test phase. They provided and evaluated a solution to estimate the effort of the test phase using various factors. Dalal and McIntosh (1994) used measures to determine when to stop testing, relying on ideas from software reliability models. Cottam et al. (2008) takes test measures as input and proposes a visualization of test results. The visualizations are highlighted as important as with the system size test diversity, and the number of tests increases. The visualization benefits have been highlighted, such as timely updates of test status based on test data. Practitioners provided positive feedback and found the visualization to be a valuable addition to their current systems.

With the increase in size and complexity, automation and tools have become important. Liu and Mei (2014) developed a platform for automation. Their framework encompasses various test activities, from management to execution and analysis, including the following modules: test initialization, test scope identification, data management, test procedure management, automated test, and test data analysis. Given the high number of tests to be managed and scheduled (Li et al., 2006), propose a unit testing framework relying on grid technology. The framework is built on a layer of specific grid services. The researchers compared alternatives for scheduling the grid resources. Using swarm intelligence for scheduling worked well concerning average task completion time.

2.4. BDD studies

This section lists the related work for the usage of BDD in software development processes. This study’s main objective is to utilize the BDD-based development process in the context of large-scale software development. In the research literature, very few studies have used BDD practices in large-scale development. We performed an informal literature review and found that most previous studies only use BDD practices in one of the phases of the software development process, as shown in Table 1.

A BDD inspired development technique (BLDD) is proposed and evaluated by de Carvalho et al. (2013). The evaluation of

Table 1
Summary of the related work.

Study	Objective	Impacted phase	Industrial evaluation
de Carvalho et al. (2013)	To provide traceability of requirements through BDD test cases.	Requirements	No
Cisneros et al. (2018)	To identify the impact of BDD on Quality attributes	Development	No
Diepenbeck et al. (2012)	To provide a new development flow, using the BDD, in context of hardware design and verification	Complete life-cycle	No
Soeken et al. (2012)	To reuse existing test steps for writing new BDD test steps.	Development	No
Carrera et al. (2014)	To propose and evaluate development of BDD based development methodology.	Complete life-cycle	Yes
Rahman and Gao (2015)	To propose an architecture that supports the reuse of BDD scenarios and test-steps	Complete life-cycle	No
Binamungu et al. (2018)	To propose approach that detects duplication in BDD documents.	Requirements	No
Häser et al. (2016)	To identify domain aware language supports in creating better BDD scenarios	Requirements	No
Lübke and van Lessen (2016)	To combine BDD with a business process model to facilitate test automation.	Testing	Yes

this approach was conducted on open source software (ERP5). The evaluation describes the mapping of "Given" "When" and "Then" to the business processes. Cisneros et al. (2018) analyzed the impact of BDD on external code quality, internal code quality, and productivity. The experiment was conducted using the students for academia. Diepenbeck et al. (2012) proposed a new development approach using the BDD in the context of hardware design and verification. The proposed approach starts with "Acceptance Tests" written before the development phase starts. These tests incorporate the quality attributes of hardware systems. Later, these tests are used to develop and verify the implementation. Soeken et al. (2012) proposed an approach that is used to generate the test case steps and their definitions semi-automatically. A user enters information in the tool and gets the recommendations to relate to the test steps and definitions. Binamungu et al. (2018) proposed an approach that detects duplication in BDD documents. Rocha et al. (2019) suggested a method that predicts the code changes using the BDD (Cucumber) test cases. The method was implemented using a tool and evaluated using 18 open source projects.

Carrera et al. (2014) proposed and evaluated a BDD based development framework. The proposed framework helps generate test-cases from the BDD scenarios and mocks to help during the implementation of scenarios. It consists of four phases: (i) writing the expected behavior, (ii) writing scenarios, (iii) breaking down scenarios, and (iv) to use the scenarios. Rahman and Gao (2015) proposed an architecture that supports the reuse of BDD scenarios and test-steps. Their study introduces the reuse of BDD and the test-steps in the context of micro-service. The approach was described, but the evaluation was missing relevant example was not detailed enough to understand the working of the approach. Häser et al. (2016) proposed a domain aware language supports to create better BDD scenarios. An experiment with groups of a student demonstrating that group using business-aware domain language performed better.

We found only three studies (present in Table 1) that describe the application of BDD on a complete-life cycle, and only two of these studies were evaluated in the industry. These two industrial studies were not applied in a large-scale context, or the context was not clearly defined in the study. A summary of previous related studies that describe the usage of BDD for improving the software development process are listed in Table 1. From the related work, we have found that there is a need to identify the benefits and challenges of using BDD in a broad context and provide a solution to address these challenges. The characteristics of the studies are present in Table 1.

Binamungu et al. (2018) identified challenges and benefits related to BDD by surveying the software practitioners. According to their study, the most critical challenges associated with BDD are (i) challenging to collaborate as it requires participants from different phases, (ii) lack of coaching (iii) template to write BDD specifications (iv) maintenance. The significant perceived benefits of BDD, according to the participants of the survey, were (i) improved test-ability of requirements, (ii) improved documentation, (iii) better capturing of the domain knowledge, (iv) better understanding and implementation of the software. It is important to note that the benefits mentioned earlier and challenges are listed without any details of the context in which practitioners are working, i.e., small organizations or large-scale organizations. Lübek and Lessen describe an experience from a large-scale software project where BDD was combined with a business process model to facilitate test automation activities. The approach helped in using requirements modeled as a business process model by the business analysts to be used as BDD-based test scripts. In this study, we try to address this research gap using BDD in a large-scale context, enlisting benefits, and its challenges.

In summary, existing literature shows that:

- Previous research has not described the benefits and challenges of using BDD in large-scale product development. There is a need to evaluate the benefits and challenges of BDD in a large-scale context
- BDD was not proposed for large-scale projects. However, the benefits BDD offer can help in addressing the challenges of large-scale projects (collaboration, communication, requirements elaboration, and verification).

3. Research approach

Our research approach is inspired by the technology transfer model proposed by Gorschek et al. (2006). The study starts with identifying improvement areas using the existing development process and other means of information (such as documentation, discussions, and observation of current practices as per Lethbridge et al. (2005)). Next, BDD is decided as a means to address the improvement areas. We start with the identification of benefits and challenges of BDD in a large-scale context, using workshops. In the next step, using information collected during the workshops, we proposed a BDD based process for large-scale development. The parts of the BDD based process are implemented and demoed in the laboratory setting. Later, evaluation of the process is performed with software practitioners' help using industrial surveys and interviews as the data collection methods. Based on the feedback from practitioners, the proposed process was improved. The details of each step are described below.

The research approach used in this study is shown in Fig. 1.

3.1. Step 1: Identify potential improvement areas based on industry needs

During this step, we assessed the existing development process, the unit of analysis, and the industrial settings. The context of industrial studies can help in understanding the applicability and results of the studies (Petersen and Wohlin, 2009). This study

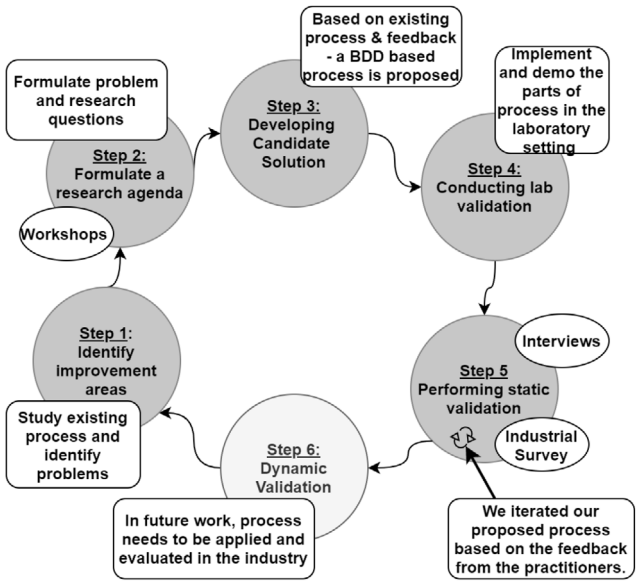

Fig. 1. The research approach used in this study.

took place in one of the product development units (developing Business Support Systems) of an organization (i.e., Ericsson) that develops a large-scale distributed product, a combination of 12 sub-products. This organization is considered one of the biggest telecommunication companies globally (i.e., more than a hundred thousand employees), and the development model is a hybrid of bespoke and market-driven development. The development teams are spread across different sites (Sweden, Germany, Canada, India) and collaborate using software tools (git, mingle, etc.). The requirements, development, testing, and delivery organizations collaborate to deliver the product in time. This study's participants were experienced product managers, requirement engineers, architects, managers, developers, testers, and service delivery practitioners.

The unit of analysis is the development process of the organization. In the existing development process, product development occurs using a hybrid model of the waterfall and agile practices. The process is divided into three different phases, requirements, development, and verification. In the requirements phase, a requirements engineer develops software requirements in system-level use-cases called business use-cases. These use-cases are written in word documents following a defined template. The vocabulary used by the requirement engineers is often not consistent between different requirement documents. The large-scale product consisted of more than a million lines of product code with 21 sub-products developed by more than 30 development teams. The majority of the participants used in the study were part of this large-scale product.

These business use-cases (system-level) are then broken into sub-products business use-cases. The documents containing sub-products business use-cases are not written by the author of business use-cases, often resulting in misinterpretation of requirements. In the development phase, product teams use agile practices such as stand-up meetings, test-first development, sprint-based development, etc., and commits to their delivery dates, regardless of other dependent sub-products. These development teams use these sub-product use-cases as software requirements. When all the sub-products are ready, these products are sent to the test organization for verification. The test organization prepares test-cases from system-level use-cases and is based on their domain knowledge. When all the manual test-cases are verified, then the product is released.

We identified that the automated test-cases are written using Java, and the manual test-cases are written in natural language, describing each test-step in human-readable form. There is a loss of information when tests are written in code from natural language requirements. The test-case development does not explicitly present the product manager's input or the customer's presence. The test-cases' quality and coverage are dependent on the test developer as requirement engineers, or the product managers do not review these test-cases. This lack of input from requirement engineers often results in missed verification of important functionalities required by the customer or spending effort on verifying functionality not needed by the customer. The existing development process of a use-case is shown in Fig. 2. Fig. 2 shows only the parts of the development process that are deemed necessary in the context of using BDD. Once a use-case is released, any defects on the use-case are handled in the maintenance phase while new use-cases are developed using the process shown in Fig. 2. These lessons were learned after several discussions with practitioners working with the process.

3.2. Step 2: Formulate a research agenda

BDD is used to overcome the issue with missed requirements during test automation and lack of input from the business perspective (Solis and Wang, 2011). Our work starts by identifying the strengths and weaknesses of BDD in the context of large-scale product development. In the next step, we used BDD's strengths and weaknesses to propose a BDD-based process suitable for large-scale software development. We formulated the following research questions to evaluate the challenges and benefits of using BDD in large-scale product development.

- **RQ 1:** What are the benefits practitioners associated with BDD in large-scale software projects?
- **RQ 2:** What are the challenges that BDD leads to in large-scale software projects?
- **RQ 3:** How can BDD be applied in large-scale software development projects?
  The RQ 3 is further divided into two sub-questions, provided below.
  - **RQ 3.1:** What are the activities needed for BDD to be applied in large-scale software development projects?
  - **RQ 3.2:** What conclusions (concerning the significance, limitation, and completeness) can be drawn during the industrial evaluation of the proposed process?

To get the software practitioners' views on the usefulness and challenges of BDD (RQ 1 and RQ 2) in large-scale product development, we have used one-hour long workshop sessions as a qualitative research method. Workshops are planned meetings to obtain the views of the group members on a specific topic. In the software engineering context, workshops have been used to get feedback from the practitioners or subject matter experts.

To increase the validity of the workshop sessions, we performed multiple tasks such as "peer debriefing", "member checking", and systematic data collection (details described in Section 6). Following steps were used as a primary source for designing and execution of all workshop, as described below:

Defining the research problem: In our workshop sessions, the objective was to identify issues related to the introduction of BDD in product development.

Planning the workshop event: We planned six workshop sessions, and each of these sessions lasted for around one hour. The workshop's agenda was shared with the participants before the start of the meeting, in the form of details of the meeting requests.

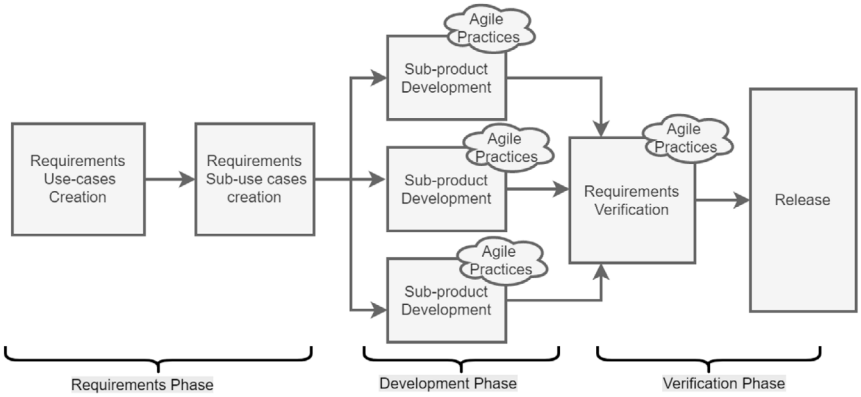

Fig. 2. Existing software development process of a use-case in the organization.

Selecting the participants: To increase the value of workshop sessions, we decided to use only the experienced practitioners (more than five years of experience) who have worked on large-scale software systems. These selected practitioners had experience with software development, requirements, or product management, and they were from three geographical sites. These practitioners understood BDD concepts, and the participants understood test-first methodologies (e.g., TDD or have used BDD). These practitioners were suitable subjects (i.e., experienced in test-first methods and from large-scale software contexts) to evaluate and identify the benefits and challenges of using BDD.

Conducting the workshop session: Each workshop session started with background information on BDD, followed by a discussion on BDD. Lastly, the discussion focused on the benefits and limitations of BDD concerning the current context. These three parts are described below:

Background Information: The purpose of this part was to share the information about BDD with practitioners. Each session started with a description of the topic by the moderator and background information on the BDD. This information was presented using the presentation slides (also shared with participants after the session). The first half of the workshop introduced BDD's objectives, the concepts related to BDD, and the implication of the concepts on current ways of working. An example of BDD was also shown to the participants to help them understand the concepts and usage.

Discussion on BDD: In this part, the practitioners discussed their understanding of the BDD. They inquired about various aspects of BDD (such as requirement handling, configuration management, tool-support, etc.) Each participant shared their views on BDD and how they see BDD usage in large-scale product development.

Discussion on current context: Practitioners were asked to reflect on BDD's benefits and limitations concerning existing ways of working. The notes were taken for each discussion item, and the moderator helped keep the discussions relevant to BDD. In the end, each participant was asked if they have any questions or feedback about the workshop's topic. Table 2 provides details of each session.

The workshop notes contained anecdotes and statements by the practitioners written on notebook pages during the session. The summary of each session describing the relevant considerations was sent to the workshop participants as the validation process. The collected qualitative data from the workshops were analyzed using the "constant comparison" approach for the qualitative analysis phase. Constant comparison method can help identify common themes and topics of the data and compare similar themes from other qualitative data. Adolph et al. (2011) described the guidelines for the constant comparison, and these guidelines were used in the data analysis. From the workshop's notes, the aim was to identify all the terminologies that practitioners used when describing BDD characteristics. These terminologies were then listed in groups of benefits and challenges. The duplicate terms were removed, and the remaining terms were used to generate results. The results of this step are described in Section 4.1.

Table 2
Detail of the workshop participants.

Workshop	Date	No. of participants	Roles of participants
W1	02/09/19	17	Developers, Test developers, Test managers, Requirements managers, Development managers
W2	11/09/19	11	Developers, Test developers, Development managers
W3	23/09/19	7	Product managers, Requirements managers, Process owners
W4	25/09/19	5	Requirements managers
W5	27/09/19	4	Test managers
W6	15/10/19	8	Test developers, Test managers

The details on the data collection and analysis of the RQ 3 (RQ3.1 and RQ 3.2) are provided in Sections 3.3 and 3.5.

3.3. Step 3: Developing a candidate solution (RQ3.1)

After identifying the key challenges, the next step was to propose a method to address these challenges.

To support the use of BDD in large-scale projects, we developed a process. To do so, we accounted for activities in the existing development process, the activities in a BDD process, the benefits and challenges identified in our investigation. The software practitioners suggested that BDD should be applied to the whole development cycle (i.e., Challenge C5) for BDD to be useful for the large-scale organization. It was also suggested that good practices from the existing development process should be utilized in the new BDD based process. BDD's original process inspires the proposed development process (in Fig. 3) and the existing development process used in the organization, discussed in Section 3.

The original BDD process assumes that BDD might help the unit testing, where the scope of test cases is limited to testing small functionality (North, 2006a). In the unit test cases, the

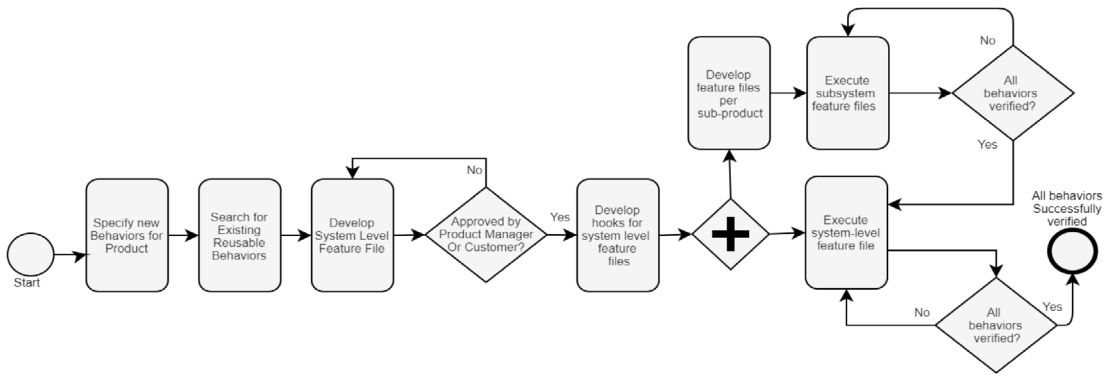

Fig. 3. A BDD based process for large-scale software projects.

objective is to verify that the product's small functionality is working fine. It also assumes that there is only one behavior (or feature) file to develop the scenarios. We proposed changes to adapt BDD for large-scale products, such as developing a system-level feature file, using sub-feature files for sub-products of large-scale products. We also adapted the BDD to incorporate the requirements and verification phases of the software development life-cycle.

The use of existing activities, tools, and practices can help mitigate a lack of competence and the scale of software development. Keeping in view these challenges, we have proposed a development process (and the associated software artifacts) that can help overcome the challenges related to BDD's usefulness in large-scale software development organizations. A significant difference in the existing development process is related to an earlier automated test case development phase. The automated test-case development phase can take place in parallel to the software development phase when using the process proposed in Fig. 3.

The details of the activities and the expected input, output and responsible actor is provided in Section 4.3.

3.4. Step 4: Conducting lab validation

During this step, the activities of the process are evaluated in the experimental setting (Gorschek et al., 2006). The lab validation started with demonstrating two cases developed using the part of the proposed process's activities. The two industrial cases were selected after discussion with software practitioners. These two behaviors (features) described aspects of the large-scale system in which the practitioners worked. The first behavior read the balance and personal information of a customer configured in the large-scale telecommunication system. The second behavior created a customer and set the new customer with appropriate settings, e.g., contract, phone number, prepaid balance, etc. The authors of this study developed these two behaviors, one each by Robot Framework (available at RobotFramework (2019)) and Behave framework (available at BehaveFramework (2019)).

Later, these behaviors were presented to the practitioners from various development phases such as product managers, requirements engineers, developers, architects, verification engineers, and development managers, demonstrating the usage of feature files, technology stack, test-case flow, key-words, and their association with back-end code, etc. These demonstrations were conducted in two sessions. These behaviors were executed on the organization's test environment during the demonstration to evaluate with realistic settings. These demonstrations were made during two sessions, and the authors noted feedback from the practitioners. Overall, practitioners liked the demonstration of the activities and suggested that this process works, as originally proposed, for large-scale product development. However, they showed concerns about supporting tools (already identified during the outcome of step 2 4.2).

3.5. Step 5: Performing static validation (RQ3.2)

The static validation step consists of collecting feedback from software practitioners using different means such as seminars, presentations, survey questionnaires, and interviews (Gorschek et al., 2006). Petersen and Wohlin (2011) suggested that when developing a solution for industry, feedback from practitioners could help identify the usefulness and drawbacks of the solution before the solution is applied in the industry. During our investigation, we initially conducted a questionnaire-based survey (available at Questionnaire (2019)) and improved our solution based on the feedback. Later, interviews of software practitioners (from Ericsson) were performed to further evaluate and improve the proposed process. The details of these two evaluations are provided below.

3.5.1. Survey questionnaire

In the first step of industrial evaluation, a questionnaire (available at Questionnaire (2019)) was developed containing four parts: (a) background on behavior-driven development; (b) description of our proposed process; (c) an example of developing 'A login system' using our proposed process, and (d) a questionnaire to evaluate the process. Molléri et al. (2020) provided a checklist to perform surveys in software engineering. This checklist was utilized during the development of the questionnaire and data analysis of the responses. As per the checklist, the questionnaire's design is classified as 'self-administrated', i.e., online form. The questionnaire contained free text input-boxes for responses that allow users to write a detailed answer to the questions. The questionnaire attempted to evaluate the significance, limitations, and completeness of the proposed process. Following questions were part of the questionnaire:

Significance: In your opinion, what are the benefits of using the proposed process?

Limitations: What are the drawbacks/limitations of the process? How can we improve the process?

Completeness: (i) 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? (ii) Are there any other steps that should be added to the process?

This questionnaire was sent to 30 practitioners of five large-scale software organizations (including the organization where the process was developed). A criterion (stated below) was designed to involve only the subjects that are relevant for this study.

Table 3
Number of survey responses from industry practitioners.

Role	No of respondents
Requirements managers	2
Requirements engineers	1
Architects	1
Developers	8
Test developers	2
Development managers	2
Total	18

- The participant has worked with large-scale software systems.
- The participant should be an experienced professional in software development (i.e., more than five years of experience).
- The participant should have an understanding of test-first development methodologies.

Eighteen practitioners (from five large-scale organizations) responded to the questionnaire, and the feedback from these practitioners was analyzed and later used to improve the proposed process. The non-respondent practitioners were sent en email as a reminder to get the maximum replies. The roles and number of respondents are shown in Table 3.

Data analysis was performed systematically to reliably identify the results and interpret the feedback from the questionnaire. An excel sheet contained the answers to the questionnaire, and the data analysis phase used this sheet. The collected qualitative data from the responses were analyzed using the "constant comparison" approach for the qualitative analysis phase (Glaser et al., 1968). During the data analysis, the responses were broken into themes related to challenges, benefits of using BDD, and feedback on the proposed process. The terminologies belonging to themes were categorized. The results of the analysis are present in Section 4.

3.5.2. Practitioners interviews

In the next step, we interviewed experienced practitioners to identify BDD's perceived benefits and challenges in large-scale development and evaluate the proposed process. These interviews helped in increasing the validity and credibility of the results identified in this study.

As per Robson and McCartan's classification (Robson and McCartan, 2016), we conducted semi-structured interviews with predetermined open-ended questions. The interview guide consisted of three parts: (i) introduction of study, interviewer and interviewee, (ii) introduction of BDD and evaluation of benefits and challenges in a large-scale context, and (iii) evaluation of the process and its activities. The objectives of conducting interviews were to:

- identify strengths and weaknesses of BDD.
- evaluate the significance, limitations, and completeness of the proposed process.
- identify activities in the proposed process that do not exist in the current development process used in the organization.
- get feedback on time to apply the process in the organization.

The interview guide (available at Guide (2020)) was designed by one of the authors of this study. Later, another author reviewed the interview guide and suggested changes to the guide. These changes were made, and the interview guide was piloted on an industry practitioner. The interview guide (available at Guide (2020)) helped in evaluating each activity present in the process by asking interviewees questions such as:

Table 4
Details of interviewees and interview.

Role	Experience	Role	Setup	Time taken
Interviewee 1	12 years	Test engineer	In-person	65 min
Interviewee 2	17 years	Requirements engineer	In-person	57 min
Interviewee 3	7 years	Developer	In-person	63 min
Interviewee 4	14 years	Test engineer	In-person	65 min
Interviewee 5	8 years	Developer	In-Person	50 min
Interviewee 6	10 years	Test engineer	Online	73 min
Interviewee 7	8 years	Requirements engineer	In-Person	75 min

- Do you currently apply the activity? If so, how?
- Would you make modifications? If so, what would you modify, and why?
- Reflections on the activity, and what alternative solutions do they see to realize the activity?
- How can we improve this activity?
- What are your views on difficulty level when using this process vs. what is already present today?

These questions (and other questions in the interview guide) were developed to evaluate the process in detail and get feedback from the practitioners to improve the process. For each part of the process, the practitioners were asked to map the activity on their current context and provide a similarity level. The practitioners were asked to suggest any changes in the process that can improve the process.

We interviewed seven experienced software practitioners who have worked in large-scale software development and understand the test-first development methodologies. These practitioners were from the same organization (and from the same site) where this study was conducted (i.e., Ericsson). Table 4 provides information on the background of the practitioners.

The interview sessions lasted between 50 and 75 min. The interviews were recorded (with permission), and during the interview, notes were taken by the author conducting the interview. The data collected from the interviews were transcribed to identify important and irrelevant information. The data analysis on the qualitative data (interview) was performed using thematic analysis as described by Braun and Clarke. The thematic analysis helps in identifying and reporting patterns within the data (Clarke et al., 2015). The relevant quotes were coded into short sentences. Later these codes were used to identify the themes. An example of this process:

- **Data:** "I think this provides a good way to describe requirements before the development organization accepts these."
- **Code:** "good way to write requirements before development starts"
- **Theme:** "Improved quality of requirements"

After analyzing the data and extracting useful information, we sent an email to each interviewee listing the interview's corresponding results. They were asked to re-confirm that correct information is extracted from the raw data of the interview. Section 4 contains the results from the interviews of the practitioners.

3.6. Step 6: Dynamic validation

During the dynamic validation, the process is piloted in industry settings. In our investigation, some activities (see details in Step 4) are already validated in the industry settings during candidate design; however, the application of complete process in the industry is part of our future work. This requires extensive

Table 5
Identified benefits per workshop and interview.

Method	Identified benefits
Workshop 1	Understanding of business aspect of requirements, Improved quality of requirements, Guide to system level use-cases, Reuse of artifacts in large-scale projects, Help for test organization.
Workshop 2	Understanding of business aspect of requirements, Improved quality of requirements, Reuse of artifacts in large-scale projects.
Workshop 3	Improved quality of requirements, Guide to system level use-cases.
Workshop 4	Improved quality of requirements, Reuse of artifacts in large-scale projects.
Workshop 5	Improved quality of requirements, Guide to system level use-cases, Help for test organization.
Workshop 6	Improved quality of requirements, Guide to system level use-cases, Help for test organization.
Interview 1	Guide to system level use-cases, Help for test organization.
Interview 2	Improved quality of requirements.
Interview 3	Improved quality of requirements, Understanding of business aspect of requirements.
Interview 4	Improved quality of requirements, Guide to system level use-cases.
Interview 5	Guide to system level use-cases.
Interview 6	Improved quality of requirements, Guide to system level use-cases, Help for test organization.
Interview 7	Guide to system level use-cases, Understanding of business aspect of requirements.

resources and long term commitment from the management before the dynamic validation is conducted. In the future, we want to focus on the dynamic validation of the activities.

4. Results

In this section, we present the results of our investigation are organized per the research question.

4.1. RQ1: What are the benefits practitioners associated with BDD in large-scale software projects?

We identified five main benefits related to the use of BDD in large-scale software projects. These benefits are; understanding of a business aspect of requirements, improved quality of requirements, a guide to system-level use-cases, reuse of artifacts in large-scale projects, and help for test organization. The benefits identified per workshop session are shown in Table 5. We provided more details about the benefit in the subsections below.

4.1.1. Understanding of business aspect of requirements (B1)

Participants of our investigation suggest that by having requirements in the form of the executable test case, from the start of the development process, a uniform and clear understanding of requirements from the business aspect can be achieved. A uniform understanding can help in reducing the re-work required due to requirements' misunderstandings.

Previous research has also associated this as a benefit of BDD, where test cases are expressed in developing a uniform understanding of requirements (Lethbridge et al., 2003). According to one of our investigation participants, this alignment on requirements between product managers, developers, and testers can help in collaboration and improved the quality of the product.

4.1.2. Improved quality of requirements (B2)

Participants of our investigation believe that BDD may help improve the quality of requirements documents in large-scale projects. By writing the requirements in the form of test cases, the end user's perspective and software requirements can be combined in the same test case.

According to one of our investigation participants, the format of the BDD test case, using **Given, When and Then** can help isolate the preconditions, business logic, and post-conditions of requirements, thus improving the quality of requirements. This template can help in getting detailed product requirements from product managers.

4.1.3. Guide to system level use-cases (B3)

Participants of our investigation agreed that the BDD tests' textual and meaningful nature makes these a suitable way to understand the system's expected behavior. They suggested that often their developers find it hard to understand the system level use-case since they are working with just a small part of the use-cases.

With the usage of BDD test cases, a developer can easily understand the system level use-case. According to one participant, these tests can help identify the priority of tasks, i.e., where to spend the effort, since we know what we need to deliver for the use case to work correctly. This ease of understanding can help the developers and testers perform their work in a better way since many of the new developers and new testers depend on their experience to understand the use-cases of software products. A participant in the workshop also mentioned that these system-level test cases could be shipped to the customer, which is a significant competitive advantage for the organization.

4.1.4. Reuse of artifacts in large-scale projects (B4)

In large-scale projects, it can be time-consuming and costly to develop each new artifact from scratch. The behavior (or features) described in BDD are useful artifacts for the development and verification phases. Practitioners suggested that these behaviors can be reused by the test organization when developing automated test cases. Furthermore, the behaviors can also be reused to serve the purpose of the product's documentation.

4.1.5. Help for test organization (B5)

The test automation developers, in workshops, revealed that the test organization is already developing BDD like scenarios. However, these scenarios are written at the end of the project, i.e., during the verification phase. With the introduction of BDD, where requirements are in the form of high-level test cases, their test automation can start as early as the development phase's starting time. The time saved at the end of the development cycle can be added to the exploratory testing, thus increasing the product's quality. Furthermore, participants of one workshop suggested that BDD's benefits can increase further if the BDD test cases are complemented with a data-driven approach. In another session, participants suggested that test organization can save costs related to automation and tool development if the whole development cycle uses a BDD process from requirements to the verification.

4.2. RQ2: What are the challenges that BDD leads to in large-scale software projects?

We identified seven challenges: the scale of the software projects, ownership, lack of Competence, cost benefits, specification of behaviors in large-scale projects, difficulty writing system-level test-cases, and versioning control of behaviors. The challenges identified per workshop session are shown in Table 6. We provide more details about these challenges in the subsections below.

Table 6
Identified challenges per workshop and interview.

Method	Identified challenges
Workshop 1	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects, Adoption of new tools and technologies, Cost Benefits of BDD in large-scale.
Workshop 2	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects, Difficulty in writing system-level test-cases.
Workshop 3	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects, Cost Benefits of BDD in large-scale, Versioning Control of behaviors.
Workshop 4	Specification of behaviors in Large-scale projects, Scale of the software projects, Difficulty in writing system-level test-cases.
Workshop 5	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects, Difficulty in writing system-level test-cases.
Workshop 6	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects, Difficulty in writing system-level test-cases.
Interview 1	Specification of behaviors in Large-scale projects, Adoption of new tools and technologies, Cost Benefits of BDD in large-scale.
Interview 2	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects.
Interview 3	Specification of behaviors in Large-scale projects, Ownership and maintenance of behaviors in large-scale projects.
Interview 4	Adoption of new tools and technologies, Cost Benefits of BDD in large-scale.
Interview 5	Difficulty in writing system-level test-cases, Specification of behaviors in Large-scale projects.
Interview 6	Scale of the software projects, Cost Benefits of BDD in large-scale.
Interview 7	Specification of behaviors in Large-scale projects, Adoption of new tools and technologies, Cost Benefits of BDD in large-scale.

4.2.1. Specification of behaviors in large-scale projects (C1)

A critical challenge identified during workshops relates to the specifying behaviors (as described in Section 2) for new development. According to the practitioners, in large-scale projects, the exact requirements are not known in advance, and the requirements are clarified and elaborated during the development phase. It is often the case that practitioners have a very high-level idea of a feature, and it requires many iterations with domain experts to understand how the potential users will use it. They suggested that it may be challenging to specify exact behaviors at the start of the BDD when things are not clear.

4.2.2. Ownership and maintenance of behaviors in large-scale projects (C2)

Another vital challenge pointed out by the participants is related to the BDD artifacts' ownership and the BDD process. There may be a lack of ownership when it comes to BDD since all software development phases share these artifacts and processes. This can result in confusion, a lack of guidance for the practitioners using the process, and a lack of maintenance support for these artifacts. Furthermore, the ubiquitous nature of BDD test cases can result in maintenance-related challenges as well. Participants of the workshops believed that the maintenance of these tests could be a complicated process because of the textual nature of test cases and corresponding code implementation.

4.2.3. Adoption of new tools and technologies (C3)

In large-scale projects, the introduction of new tools and technologies can be difficult and time-consuming. The practitioners may require training and education to be productive in a BDD setup. Practitioners belonging to the test organization recommended that a proof of concept be conducted to evaluate BDD tools' suitability for current product interfaces. Lack of familiarity with the new tools supporting BDD is identified as a challenge that impacts BDD adaptation in a large-scale project.

4.2.4. Cost benefits of BDD in large-scale (C4)

According to the practitioners, BDD's process and artifacts can be costly to develop in large-scale projects. The analysis and modeling of BDD test cases, in a large-scale context, can take more time than the ordinary requirement documents. Modeling the BDD test-cases can require rigorous analysis and adequate system knowledge from a practitioner. The development of test-cases involves collaboration between stakeholders like requirements, development, and test organizations. All of this collaboration can result in higher development costs and time. Practitioners also mentioned that the organization had invested a lot in the existing test framework, and replacing that test framework with a new BDD test framework may result in higher costs. The storage and retrieval of BDD test cases also require new documentation storage setup, which can also result in higher costs.

4.2.5. Scale of the software projects (C5)

Large-scale software products, where components are developed across different sites, require a great deal of communication, iterations, and inter-connected work. According to practitioners, attending the workshops can be a limiting factor for the BDD process, as BDD assumes close collaboration between different stakeholders and on-time delivery of sub-components of large-scale products. An example was provided by the practitioners, where a single-use case involves multiple components to be delivered and tested together, and often, a use case is changed at very late in the development cycle.

4.2.6. Versioning control of behaviors (C6)

Another identified challenge is the versioning control of the behaviors (as described in Section 2) in large-scale projects. With many stakeholders and the associated artifacts, it may be hard to keep track of required changes e.g., product managers realize that behavior is not entirely correct. Then this needs to be reflected in all artifacts.

4.2.7. Difficulty in writing system-level test-cases (C7)

It was also pointed out that large-scale products and projects meant that it is hard to write a system-level test case with only one use-case in mind. The use-cases are often inter-dependent on other use-cases, and verification may be stopped even if one use-case is missing. This difficulty in writing system-level test cases can be a challenge when using BDD in large-scale products.

Furthermore, it was mentioned that specifying system-level behaviors may require collaboration from various stakeholders keeping in view the needed business and technical skills. One of the workshop participants believed that BDD might increase the time to develop and deliver the behaviors in large-scale projects. Practitioners believed that requirements and test artifacts and processes have different needs, and combining these processes, as done by BDD, may not be very efficient.

4.3. RQ3: How can BDD be applied in large-scale software development projects?

This research question consists of two sub-questions describing the proposed process's activities and the industrial evaluation details. The results of the two sub-questions are described below.

4.3.1. RQ3.1: What are the activities needed for BDD to be applied in large-scale software development projects?

The process's activities are described below, and the input/output of each activity is described in Table 7. Table 7 contains the name of the activity, the actor acting, and the expected input and output of the activity. Each activity in the process requires an input and produces an output for the next activity. Each of these activities has defined actors responsible for performing the tasks needed to fulfill that activity, e.g., a requirements engineer working with stakeholders (testers, domain experts, etc.) to complete a system-level feature file.

Specify new Behaviors for Product: The product manager, acting on customer request, write new behaviors (also called features, described in Section 2) to be added to the product. These new behaviors can be in the form of textual statements or short descriptions.

Search for Existing Reusable Behaviors: During this activity, a requirements engineer searchers for existing similar behaviors that can be adapted and reused. This activity is needed to avoid developing and testing behaviors when similar behaviors are already present. This can lead to the reuse of the features and tests. The presence of a similar feature means that development and testing phases can also reuse existing artifacts. A few methods to conduct the searching for reusable behaviors are provided below:

- In a large-scale organization, document searching is commonly used by the practitioners to identify reusable artifacts.
- Text classification approaches used in natural language processing can help identify reusable behaviors (see, for example, Zaïane and Antonie (2002)).
- Normalized Compression Distance: The similar artifacts are identified using a script based on normalized compression distance (c.f., Feldt et al. (2016)).
- Similarity Ration: Irshad and Petersen (2020) defined a similarity measure and process to identify similar reusable documents. This approach can be used to identify reusable behaviors (see details at Irshad and Petersen (2020)).
- Searching in a repository management service (e.g., Gerrit, Stash) can help identify similar behaviors.

Example: To exemplify the searching process, we can take the example of a search using the normalized compression distance. We assume that a requirement engineer must write a new BDD behavior on a feature with the description: "A login system is supported by the product". Following steps are required for this activity:

1. The requirement engineer selects the text (or keywords) of the feature's description.
2. The requirement engineer selects an automated script implementing NCD (an implementation available at Script (2020)).
3. The existing feature files are placed in a folder, each feature file having a unique name.
4. The search is performed using the feature's description or keywords. NCD works by comparing the compressed sizes of the description of new behavior and compressed size of each feature file with their concatenation's compressed size. NCD values lie between "0" and "1", where a value closer to "0" means that the similar feature file is found while a value closer to "1" represents that no similar feature file is found.
5. The requirements engineer reviews the feature files with values closer to "0" and assesses the relevance of the behaviors inside the feature files for reuse.
6. The requirement engineer identifies the closest matching behavior from the feature files.
7. The behavior is modified for reuse.

For other search techniques (such as text classification, similarity ratio), steps 2 and step 5 are changed based on the technique's implementation. The reuse process's complete details are described and evaluated in another study by the authors (available at Irshad and Petersen (2020)).

Develop System Level Feature File: In this activity, the requirements engineer, elaborates on the behaviors and adds more details (e.g., steps to perform actions, expected inputs/outputs, etc.). A system-level feature file is developed that contains executable scenarios contained in a behavior. This file (containing behaviors) is approved by a customer or product managers to have an agreed executable feature file and agreed customer requirements.

Develop hooks for System Level Feature File: In this activity, test developers develop hooks (or glue code (Coveros, 2019)) for the feature files. The hooks are the implementation of keywords/sentences created in the previous step, e.g., a keyword "A login system is present for the product" is connected with test-code written in any programming language. Similarly, all keywords are linked to unique test methods written in a programming language. Once these hooks are developed, the feature file (along with hooks/glue code) keeps executing in the integration test-case environment.

Develop feature files per sub-product: The behaviors in the system-level feature file are defined for a single black-box like system. However, in this step, the behaviors (and corresponding hooks) are divided as per the division of sub-products in a large-scale system. Next, each sub-product develops the product code against the features files belonging to their sub-product. The product code is tested using the sub-feature file.

Execute sub-system feature files: This activity belongs to the sub-products that are developing their corresponding sub-feature files. This activity helps the developing team of the sub-product to identify when their new behaviors are complete. When the corresponding behaviors are ready, the behaviors' code is sent for system-level verification (see next activity).

Execute system-level feature file: The system-level feature files execute the behaviors written during the phase "Develop System Level Feature File". Initially, all the behaviors may fail. When the sub-features are complete, and the behaviors are verified, the feature is considered completed and ready for release.

Example of Usage of Process in Large-scale Organization In this section, we demonstrate the use of activities of the proposed process in a large-scale context. The example is based on the use case belonging to the authors' large-scale organization. The example concerns a large-scale system having three components as described below:

- A sub-product contains all the customer-related functionality (called Product X in this example).
- A sub-product contains all the user contracts related functionality (called Product Y in this example).
- A sub-product contains all the packages offered to the customer (called Product Z in this example).

Start: A telecommunication customer requests a software development organization to develop a feature described as "A

Table 7
Description of activities in proposed process.

Activity	Actor	Input	Output
Specify new Behaviors for Product	Product manager	An idea from Product manager or requirements from customer for adding new behaviors in the product.	A product manager communicates description of behaviors expected in the product to requirements engineers.
Search for Existing Reusable Behaviors	Requirements engineers, Domain experts	Description of new behaviors from product manager is provided.	If similar existing behaviors exists in products, then these are identified and adapted for new cases.
Develop System Level Feature File	Requirements engineers, Architects, Verification engineers	Description of new behaviors from product manager is provided.	A BDD feature file containing the behaviors that may be developed. The behaviors are in the form of executable scenarios (test-cases).
Develop hooks for system level feature files	Test Developers	A BDD feature file containing the behaviors as executable test-cases.	Executable keywords and the back-end implementation to make the behaviors executable as automated test cases.
Develop feature files per sub-product	Architects and Developers	Executable keywords and BDD feature file from activity "Develop System Level Feature File".	Sub-feature file containing relevant behaviors for each sub-product and the implemented product code of the new behavior.
Execute sub-system feature file	Developers	Executable sub-feature file and corresponding product code.	A test case execution report for each sub-product and behavior.
Execute system-level feature file	Developers, Test engineers	Executable feature file containing behaviors.	A test case execution report.

customer goes to the franchise store to sign up for a data package. As an agent, orders and activates the required package for the customer."

Activity: Specify new Behaviors for Product: The product manager, acting on customer requests for a new feature, writes new behaviors for the large-scale system to fulfill the feature.

- Behavior 1: Create and activate a customer in the system.
- Behavior 2: Create and activate a contract in the system.
- Behavior 3: Start contract with details of Data Package.

Activity: Search for Existing Reusable Behaviors: For the sake of this example, we assume that no reusable behaviors are present; therefore, the behaviors need to be developed from scratch. The remaining activities are performed based on this assumption. An example of this activity is already provided in the activity description.

Activity: Develop System Level Feature File: The behaviors are elaborated and written in a system Level "Feature File". These elaborated behaviors are:

Behavior 1: Create and activate a customer in the system.
Given Send a Create Customer request with ID 1 to Product X
When Activate Customer with ID 1 in Product X
Then Verify Customer with ID 1 is active in Product X

Behavior 2: Create and activate a contract in the system
Given Send a Create contract request with ID A to Product Y
When Activate contract with Id A in Product Y
Then Verify contract with ID A is active in Product Y

Behavior 3: Start contract with details of Data Package
Given An item called Data Package is present in Product Z
When Data Package is fetched correctly
And Data Package item is added to Contract A in Product Y
Then Verify Contract is updated in Product Y

Activity: Develop hooks for system level feature files: The hooks are the implementation of keywords/sentences created in the previous step. The following examples show the hooks corresponding to keywords from Behavior 1. Similar hooks are developed for Behavior 2 and Behavior 3 as well.

```
@Given('Send a Create Customer request with ID 1 to Product X')
def create_customer_product_X():
    # Code that sends the request to create customer in Product X

@When('When Activate Customer with ID 1 in Product X')
def activate_customer_product_x():
    # Code that activates the customer  in Product X.

@Then('Verify Customer with ID 1 is active in Product X')
def verify_customer_active():
    # Code that verifies customer is active in Product X
```

Activity: Develop Feature Files Per sub-product: In our example, the behaviors are based on (1) Customer related functionality from Sub-product X, (2) Contract related functionality from Sub-product Y, (3) Package related functionality from Sub-product Z. These three correspond to three different sub-systems; hence the system-level behaviors are divided into three sub-system feature files as described below:

- Sub-feature file 1: Contains the elaborated Behavior 1 and hooks used by Product X's development teams. The development team uses the sub-feature file 1 as a requirement and test document to develop the product code for Behavior 1.
- Sub-feature file 2: Contains the elaborated Behavior 2 and hooks used by Product Y development teams. The development team uses the sub-feature file 1 as a requirement and test document to develop the product code for Behavior 2.
- Sub-feature file 3: Contains the elaborated Behavior 3 and hooks used by Product Z's development teams. The development team uses the sub-feature file 1 as a requirement and test document to develop the product code for Behavior 3.

The development teams then develop product code to fulfill the requirements present in their corresponding sub-feature file, e.g., Behavior 1 is developed in Product X by using Sub-feature file 1.

Activity: Execute subsystem Feature Files: The automated sub-feature 1 behaviors are tested against the product code developed for Behavior 1. A test report is generated for the sub-product development teams and stakeholders.

Table 8
Significance: Summary of feedback from industrial evaluation.

Significance of using the process	Noted in
Improved communication between stakeholders	7 replies
It helps in understanding end-user needs and focus on essentials.	5 replies
It helps in the early detection of interface non-alignments between large-scale systems.	4 replies
Structured way to write requirements.	1 reply
Non-technical communication between stakeholders.	1 reply
Out of the box documentation of the whole system.	1 reply
Real-Time tracking of progress of development in distributed software systems.	1 reply
Clearly defined responsibilities.	1 reply
Supports the reuse culture in the organization.	1 reply
Involvement of customer before development starts.	1 reply
Improved quality of the product with this process.	1 reply

Table 9
Limitations: Summary of feedback from industrial evaluation.

Limitations of using the process	Noted in
Dependence on far too many stakeholders.	8 replies
Behaviors may not be easy to define for large-scale systems with many dependencies and requirements e.g., non-functional requirements	3 reply
It requires learning of new technologies.	2 replies
Customer may not be interested in Feature file iterations.	1 reply
Lesser involvement of Architects as compared to Product managers.	1 reply
Time-consuming and costly process.	1 reply
Difficult to develop sub-feature files.	1 reply
Maintenance of test-cases can be a problem.	1 reply
No support for the delivery of partially complete solution.	1 reply
No drawbacks mentioned by practitioners	6 replies

Activity: Execute System Level Feature Files: The execution of a system-level feature file generates reports describing each keyword's status, e.g., if keywords of Behavior 1 are passing or failing. An example report containing results from Behavior 1 is shown in Fig. 4.

4.3.2. RQ3.2: What conclusions (concerning the significance, limitation, and completeness) can be drawn during the industrial evaluation of the proposed process?

The proposed process and its components were evaluated with industrial practitioners' help, and their feedback helped improve the process. In general, practitioners were optimistic about the proposed process and suggested four changes to improve the proposed process. The results of each aspect of evaluation (significance, limitation, and completeness) are provided below.

Significance of using the process: The results from the industrial evaluation show that practitioners believe that several benefits are associated with this process, such as improved communication and collaboration between stakeholders for technical as well as non-technical communication. They believe that the proposed process enhances communication by involving stakeholders in discussions related to the feature files. They also suggested that the system-level feature file can develop a better end-to-end understanding of the product's use-cases. Few practitioners suggested that the process may result in early detection of the misaligned interfaces of the sub-product. A practitioner with a background in requirements engineering believed that the structured way of writing scenarios and behaviors might improve the quality of the requirements and provide out-of-box documentation of a large-scale product's behavior. The recommendation on the actors for each activity was also considered a benefit of this process as this defines and delegates the responsibilities in a better way. The support for reuse culture and tracking of progress with executable system-level feature files were also mentioned as strengths of this process.

The significance of using the proposed process and the number of corresponding replies from industrial surveys and interviews are described in Table 8.

Some quotations from the survey respondents regarding the benefits are given below:

> "Improves communication between all stake holders and enables them to easily engage in product development cycle. And BDD focuses on the functionality more which ensures you are delivering the value business needs."

> "It can definitely solve misunderstandings between product management and development organization."

> "By defining tests and preparing for automated tests before anything is developed, a faster, and more business-driven approach of the development can be achieved. Preferably, for a contract with a customer, each paragraph/clause in the contract should be managed and validated in the same way (TDD)."

> "The main benefit is the increased focus on the business value for the user/customer. The BDD emphasize focus on 'WHAT' shall be achieved."

Limitations of using the process: During the evaluation, practitioners were asked to identify the drawbacks or weaknesses of the proposed process. According to the practitioners, the process requires collaboration between many stakeholders, which makes it a costly and time-consuming process at the start. One practitioner mentioned that it is challenging to develop a sub-feature file. The practitioners also suggested that maintenance of the feature files and the sub-feature files can be a possible drawback due to these feature files' textual nature. One practitioner described that defining behaviors, the first activity of the process, may not be straightforward. This compulsory activity of defining the behaviors at the start of the process can be considered a drawback. In rare cases, requirements are understood with the help of experimentation during development phases. One interviewee mentioned that involving customer to approve the feature file before the development starts can become a bottle-neck and delay the development. During the evaluation, a practitioner mentioned that sometimes our organization sends partial deliveries to the customer to secure milestones, and with this process, only finished features can be delivered.

Table 9 shows the limitations (and the number of replies) found during the evaluation of the process.

Some quotations from the survey respondents regarding the drawbacks are given below:

> "Can be hard to write correct testcases since nothing is implemented. Testcases for all sub-product and a "final" testcases that connects all the dots (sub-products) that finalizes the customers requirements."

> "I think in this process few people/teams need to work in-between and this might be a problem if there are not enough resources available/allocated for doing this."

> "The main Pro for BDD is also the main limitation which is availability of all stake holders. Absence of any team/member can cause ambiguities in overall process."

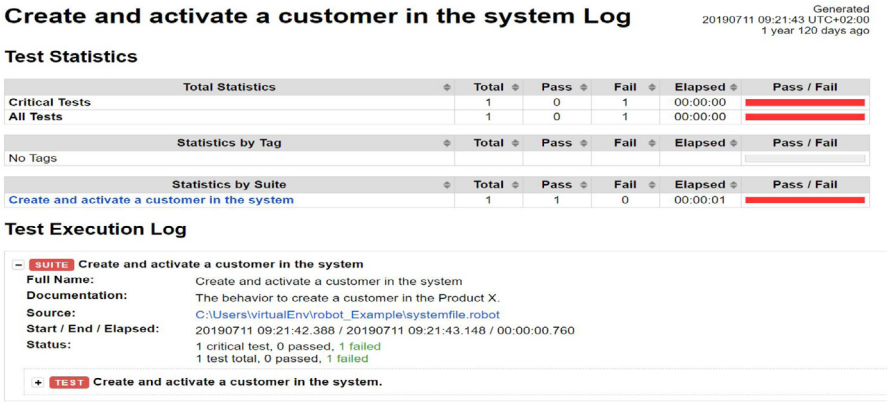

Fig. 4. An example report from the activity 'Execute System Level Feature Files'.

Completeness: Steps to be removed from the process: The practitioners were asked to identify activities that are not necessary or important for the proposed process. Only one practitioner (during the interview) suggested removing the searching for existing reusable artifacts as a mandatory activity of the process. The practitioners motivated his suggestion that this activity is often performed based on the practitioner's expertise and experience. Therefore, there are no clear guidelines present for this activity. A quotation from another practitioner is given below.

> "No, I belive that all the steps are nessesary but in time, when the process get's more established steps might be changed or removed naturally, because both dev teams and clients get more experience on handling the process."

Completeness: Steps to be added to the process: To improve the process using the practitioners' feedback, the questionnaire and interview guide asked for suggestions to add a new activity in the proposed process.

There were four leading suggestions by practitioners, and these suggestions are quoted below:

> **1. Architectural changes:** "I think that there can be a case when an architectural design problem is not found until later test phases. There could be a loop back to development to re-engineer the added feature. Maybe not the wanted behavior but sometimes a necessary evil..."

> **2. Consistent Terminologies:** "BDD requires you to be extremely stringent about your tests, keywords, format, and how they are structured. But I'd say that's a benefit as well."

> **3. Interface changes:** "Activity for approval of interfaces where two subsystem needs to communicate with each other."

> **4. Collaboration:** "Like I said above, the process works well if it is being applied to a single element/TPG of a larger and complex system. For developing complex, compound system level features and functions need a heavy co-operation, making sure everyone understands and follows the process in the same way. This is a challenge for which there are no proven steps that can be implemented."

Table 10 contains a summary of feedback related to the completeness of the process from the practitioners provided during industrial evaluation.

Table 10
Completeness: Summary of feedback from industrial evaluation.

Steps to be added to the process	Noted in
Architectural problems discovered very late in development should be handled.	10 replies
A step to verify if interfaces are defined and agreed between sub-systems.	9 replies
A step to verify if interfaces are defined and agreed between sub-systems.	2 replies
A step that verifies the conformance of feature file with organization guidelines.	1 reply
Steps to be removed from the process	**Noted in**
The Step "Search for Existing reusable Behavior" should be removed as it is not necessary for the process.	1 reply

Final version of proposed process - After the feedback

The practitioners' feedback was considered, and the proposed process was improved based on the suggestions from them. Following changes were introduced in the process:

- A new check was introduced to validate that the developed system-level feature file is according to the standard, and ontology is consistent with organizational concepts and content of previously developed feature files. This validation may help in developing a feature file consisting of already agreed keywords.
- A new activity is introduced that deals with the interface and architectural changes. In this activity, architectural changes are evaluated, and all sub-products agree on new or modified interfaces.
- To accommodate architectural changes found late in the development process, as suggested by practitioners, a check is introduced to counter such a scenario. This check may revert the process to the activity where architectural changes are evaluated.

The final version of the proposed process is shown in Fig. 5.

Lead time to apply the process

During the interviews, we asked the practitioners to estimate the time to apply this process in the organization based on their previous experience of similar changes. Interviewees described that the process requires lesser lead time related to new activities. However, the bigger change is to educate and convince the practitioners regarding the benefits of BDD based process. Practitioners

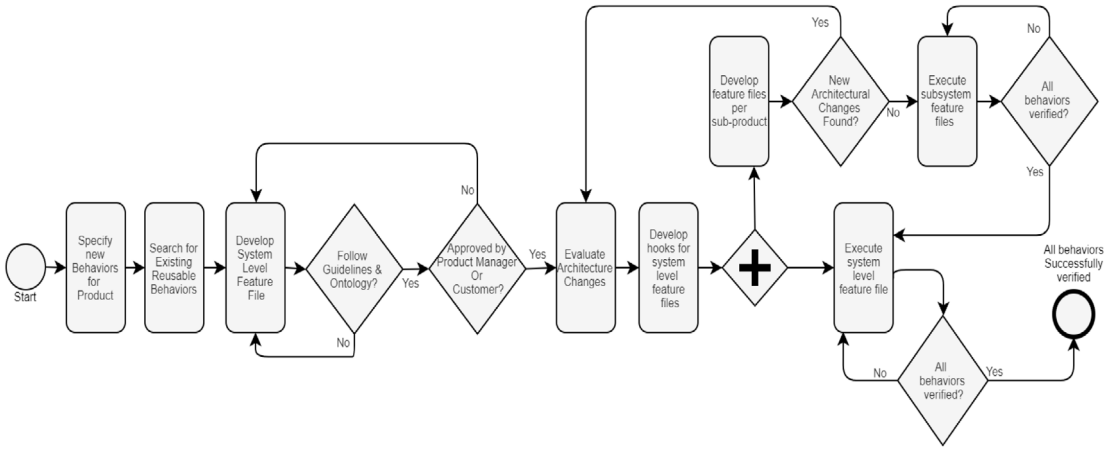

Fig. 5. Final Version: A process for large-scale product development.

estimated that it might take from 1 year to 2 years before the process is fully implemented in large-scale organizations. Table 11 provides the different estimates provided by the interviewees.

Common and different activities

During the interviews, we asked the practitioners to identify: (i) the activities that do not exist in the current development process, (ii) the activities that exist or partially exist in the current development process. We found that most activities partially exist, and the activities that do not exist are "Search for Existing Reusable Behaviors", and "Develop hooks for subsystem level feature files". Table 12 shows the reply to this question provided by each interviewee. The notation "Not Present" describes when no similar activity is present, "Partially Present" describes when the activities exist in some form (may need modifications before applying), e.g., features are written in word documents but not in feature files as process demands.

5. Discussion

This section provides a discussion on the crucial aspects of the research questions and industrial evaluation.

5.1. Characteristics of the proposed process

In Section 4, we have proposed a development process that utilizes BDD practices for large-scale software development. Essential characteristics of the process are discussed below:

System level feature file for large-scale development: The system-level feature files, developed at the start of the process, contains vocabulary and terms that are used and understood by the whole organization. A check is introduced to validate that feature files include commonly understood business needs, ontology, and vocabulary. This helps in large-scale development as different teams, and development units have less chance of misunderstanding the requirements (Solis and Wang, 2011). The involvement and approval of feature files from the product managers or customers help reduce the chances of missed requirements. Furthermore, the system-level feature file can help develop business-critical features since the feature file contains scenarios describing the business needs.

Saved effort with parallel product and test development: According to Kumar and Mishra (2016) test automation development takes around 30% of costs of software development (Kumar and Mishra, 2016). As the feature file contains executable test scenarios, this feature file can be used by product development teams as a requirement, and the system-level test organization can use the feature file as an automatic system test. In this way, product development and system-level test automation phases can be executed parallel, thus, saving time to verify and deliver the product. In our proposed process, test automation and product development can run in parallel, resulting in reducing time to deliver by 30%.

Early feedback loops: Previous research in software integration projects has shown that rapid feedback loops are essential to identify, evaluate, and fix the problems with software artifacts cheaply and quickly (Mårtensson et al., 2018). As large-scale product development is an iterative process; therefore, quick feedback loops regarding requirements, architecture, and testing can help identify and resolve the issues quickly. In our proposed process, multiple feedback loops are present, helping identify the potential problems relatively quickly. These feedback loops are on completeness of requirements, uniform vocabulary, structural misalignment, and verification of requirements, as shown in Fig. 5. The process assumes that a BDD-based use-case is only released when development is complete and sub-products are ready. However, in exceptional circumstances, when a partial delivery is needed, then gateways may be overridden, and remaining parts of the BDD use-case are released in the next iteration, thus completing the process's flow.

Less coupled team practices: The characteristics related to software teams such as the process that the team follows, the technology that the team uses to play an essential role in the success of a team (Guinan et al., 1998). Therefore, any development process must empower the teams to select the team's internal development practices and only guide the process level. In our proposed process, we have not limited the development or verification teams to use any particular tools (e.g., Jenkins, JUnit, etc.), processes (e.g., XP, pair programming), or technology (e.g., Java, Python). The process operates at an abstract level without impacting the internal process of teams. This transforms our proposed BDD based process more natural as the teams are not impacted internally.

Controlled architectural evolution: Large-scale software products are developed in multiple iterations, and each iteration results in architectural changes (evolution). Studies had shown that structural degradation might happen to a product's architecture when it evolved multiple times (Jazayeri, 2002). By introducing a separate activity that evaluates architectural changes, the proposed process provides a controlled way to evaluate and change architecture. The architects can also use this activity to agree on the interfaces and other technical details (e.g., headers in the requests, responses).

Table 11
Lead time for applying this process.

Interviewee 1	Interviewee 2	Interviewee 3	Interviewee 4	Interviewee 5	Interviewee 6	Interviewee 7
1 year	1–2 years	1–2 years	10–12 months	1–2 years	1.5 years	1 year

Table 12
Common and different activities. Partially present = ∂, Not present = ⊙.

Activity	Interviewee 1	Interviewee 2	Interviewee 3	Interviewee 4	Interviewee 5	Interviewee 6	Interviewee 7
Specify new Behaviors for Product	∂	∂	∂	∂	∂	∂	∂
Search for Existing Reusable Behaviors	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Develop System Level Feature File	∂	∂	∂	∂	∂	∂	∂
Develop hooks for subsystem level feature files	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Develop feature files per sub-product	∂	∂	∂	∂	∂	∂	∂
Execute system-level feature file	∂	∂	∂	∂	∂	∂	∂

Support for reuse: An initial activity in the process (i.e., Search for Existing Reusable Behaviors) requires that requirements engineer looks for similar behaviors that can be reused for creating a new feature. Identifying related items at the start of the process has a significant impact on the cost, quality, and development time of new features. The similarity of features may help identify similar code and similar test artifacts developed previously. However, it is essential to mention that this study's scope is limited to the BDD based process for large-scale product development, and the scope does not cover the reuse related practices such as developing for reuse, searching for reusable BDD based artifacts, etc. A separate study is planned to focus on the reuse potential of BDD based artifacts (i.e., human-readable documents) in a large-scale context similar to the work by Rahman and Gao (2015) in the context of microservices architecture.

5.2. Addressing the challenges identified by software practitioners

In Section 4, practitioners suggested challenges that prevented them from adopting BDD. The proposed process in this study attempts to mitigate a few of these challenges, as described below.

Challenge: Specification of behaviors In the new proposed process, specific actors are assigned to specify the behaviors early in the development life-cycle. Moreover, checks have been introduced to get the customers' approval of behaviors (or product managers). With the help of these two steps, the specifications of behaviors in large-scale projects can be improved. Furthermore, activity is introduced to search for similar existing behaviors, making it easier to specify the behaviors.

Challenge: Ownership and Maintenance The new process clearly defined responsibility matrix (Actors in Table 7) that can help reduce the confusion related to ownership. A product manager can be considered as the owner of the BDD related processes. However, the feature file and corresponding sub-feature files are owned and maintained by their corresponding product development teams. The maintenance-related challenge can be tackled using a similar approach as used in version control and maintenance of requirements documents. The organization already uses versioning control software.

The maintenance-related challenge to the different versions of the system-level feature file, sub-system feature file, and the hooks can be resolved with software traceability. Organizations can utilize information retrieval techniques to trace the link between different software artifacts, as demonstrated by Cleland-Huang et al. (2014). The maintenance of the system-level behaviors can be resolved by performing refactoring on the behaviors each time a new behavior is added to the system level feature file. A separate study provides an approach and guidelines on the refactoring of BDD behaviors (Irshad et al., 2020). The hooks implementing the test code are written in modern programming languages. The IDEs (such as Eclipse, IntelliJ) contains plugins that support the maintenance of BDD behaviors and hooks, e.g., when a hook is no longer used when a keyword is missing for a hook (Borg and Kropp, 2011).

Challenge: Adoption of new tools and technologies As mentioned previously, the proposed process uses existing tools and practices, and impact is least on the development team; hence there is a less likely scenario when competence is not present in the organization. Furthermore, each new change in the organization requires strategic investment from management to keep the organization competitive (Gebhart et al., 2016). For this change to happen, education and training sessions may be needed. Furthermore, an industrial evaluation showed that two use-cases were developed with relative ease, confirming that improving the competence may be easier than what practitioners initially suggested.

Challenge: Cost Benefits of BDD in large-scale To make the process less expensive, an activity was introduced in the new process that identified the reusable content already present that can be used for the new feature file. The development process may be expensive, but future benefits such as reduced defects and improved product quality may balance out the expensive process in the long term.

Challenge: Scale of software products The new proposed process keeps in view the large and distributed scale of the product. It identifies that it is important to agree on: uniformly understood system-level feature files, using commonly understood vocabulary and well-defined interfaces and architectural changes. These two steps can ensure that deviation from software requirements and architecture principles is reduced in large-scale products.

5.3. Adaptation constraints for organizations

Based on the results of the industrial evaluation, the application of the proposed process is a time-consuming and complicated task that requires an organization's long-term commitment and budget for incremental transformation. Organizations need to allocate budget and personnel to plan and transform the organization to use the proposed process. This adaptation of a new process may require competence upgrade (e.g., training for new product managers), re-organizing resources, and organizational guidelines (e.g., ontology, architectural, BDD templates). A separate and long term study is required to plan, apply, observe, and report the findings of this process's application.

5.4. Comparison with similar approaches

As mentioned in Section 2 three studies, according to informal literature review, have proposed to use BDD for the complete life-cycle of product development (Carrera et al., 2014; Diepenbeck et al., 2012). A discussion on the comparison of approaches

present in each of these three studies and our proposed approach is provided in this section.

In their study on circuit design, Diepenbeck et al. (2012) suggested a product development method that uses the BDD feature file as a first requirement document. Each step in the feature film is then developed in a single iteration using the programming language VHDL. The approach suggested that test-data (configurations) and BDD test-cases should be separated from each other to use the formal verification methods for testing. The approach is limited to small scale software product development. There are two main differences between our proposed approach and the approach proposed by Diepenbeck et al. (2012): (i) their approach is not suitable for a product with multiple sub-products since the flow assumes that only one statement may be developed each time, making it difficult to perform parallel development, secondly, (ii) the approach is not applied on any industrial case instead the demonstration of approach is conducted using a simple example. Essential aspects of development, such as reuse, system-level testing, and customer involvement, are not mentioned in the study (Diepenbeck et al., 2012).

Carrera et al. (2014) proposed a development methodology called a BEAST Methodology that is applied in the context of multi-agent systems. The method has few similarities with our proposed approach, i.e., the product owner suggests the behaviors, and later, these are elaborated and described for industrial use. However, there are some fundamental differences between our proposed approach and the approach proposed by Carrera et al. (2014), such as there is no uniformly used system-level feature file, there is no support for reuse, the development and testing are sequential processes. Our proposed approach provides a mechanism to support a common understanding of BDD in large-scale systems, which is another essential difference between these approaches.

Rahman and Gao (2015) conducted their study on BDD in a product's life-cycle in the context of microservices architecture. A micro-services architecture does not necessarily mean it involves a large-scale software organization (as defined in Section 1). This study deals with reuse and maintenance aspects of BDD test-cases in a complete product life-cycle. The study does not propose any development process for large-scale software development. Hence, our research is different concerning the context and the objective than the study by Rahman and Gao (2015). However, the reuse of BDD test-cases and their maintenance-related approaches are proposed by Rahman and Gao (2015) in combination with our proposed approach.

In short, all of these approaches and our approach are providing the solution to different problems even though these are using BDD on complete software development life-cycle.

6. Threats to validity

Runeson et al. (2012) classifies the validity threats into four types and suggests ways to improve the validity threats. These four validity threats are reliability, construct validity, internal validity, external validity. These four validity threats and measures to improve the validity are discussed in this section.

Reliability deals with the extent to which the data collection and data analysis are dependent on the researcher (Runeson et al., 2012). To minimize this threat, Runeson et al. (2012) suggested using 'peer debriefing' where more than one researcher is involved in reviewing and interpreting documents. During each workshop session, one of the researchers took notes, and after the session, these notes were discussed and finalized with the help of the second researcher. In a few cases, meeting notes (in the form of meeting minutes) were sent to the workshop participants for their feedback or clarifications. Later, an analysis was systematically performed on these notes using the constant comparison method. The results from the analysis were also discussed and agreed upon by two researchers (peer debriefing). In industrial evaluation, the data was collected in an automatic way using the web forms, reducing the chance of missing valuable information by the researcher. Later, the two researchers agreed to interpret the practitioners' feedback before using this feedback to improve the proposed method. In short, we have tried to mitigate this validity threat by involving two researchers (i.e., peer debriefing), using automatic data collection and storage (i.e., audit trail, as suggested by Runeson et al. (2012)), and using a systematic method for data analysis. To triangulate the findings, interviews were conducted to re-confirm the results from the workshops and industrial surveys. These interviews were recorded, and the findings from each interview were sent to the interviewee to increase our investigation's reliability, i.e., known as member checking as per (Runeson et al., 2012).

Internal Validity concerns the factors studied by the researchers, i.e. if these factors are affected by any other factor unknown to the researcher (Runeson et al., 2012). This threat applies to the study's part, where participants listed the benefits and challenges of adapting a BDD based process. The practitioners may identify the benefits and challenges because of other non-BDD-related factors (e.g., problems with BDD based process maybe because of other issues that the practitioner did not account for). To mitigate this threat, Runeson et al. (2012) recommended using triangulation — to use data from different sources. Therefore we have conducted multiple workshop sessions with multiple participants located at various sites. To mitigate the researcher's bias during an industrial evaluation, two researchers were involved in the preparation, writing questionnaire, and analysis of the feedback to improve validity. Similarly, the interview guide was finalized by two researchers as well, and member checking was conducted to reduce this threat to the validity of our study.

External Validity deals with the extent to which the results can be generalized, and the results are of interest for people outside the organization (Runeson et al., 2012). Even though the first part of this study took place in one software development organization, we have included industrial practitioners from four other organizations working in large-scale software products during the industrial evaluation. Furthermore, the industrial practitioners who were part of this study belonged to different roles with different experience levels, e.g., product managers, architects, developers, tester, etc. This involvement of practitioners from multiple software organizations and multiple roles may help mitigate this threat to our study's validity.

The evaluation was conducted with experienced practitioners working in different roles (e.g., developers, testers, requirement engineers) in the organization who have a prior understanding of test-first methodologies; therefore, it was challenging to select practitioners from the organization randomly. We developed selection criteria before sending the survey and interview calls to reduce the bias. These measures might have helped in reducing the potential bias, but there is a possibility that the researcher's bias exists when selecting the participants.

Construct Validity concerns with how well the study captures the construct what it intends to capture (Runeson et al., 2012). To mitigate this threat, the researchers presented BDD's background in each workshop, and participants inquired about any missed/not understood parts of the BDD. Furthermore, the researchers had considerable experience working in the company, and they were able to explain the concepts in vocabulary understood by the practitioners from the organization. Runeson et al. (2012) has described this usage of standard terms and vocabulary as a way to reduce this threat to the study's validity. For industrial evaluation, a detailed description, background, related concepts,

and example were provided to each participant to understand the concepts correctly. For the participants from outside organizations, an additional step was taken where a researcher explained to them the intended concepts behind the proposed method.

For the interview studies, we carefully designed the interview guide; we provided examples of each step in the proposed process so that the interviewee can understand all the details. A presentation was made to the interviewees on the basic concepts and working of BDD as well. Thus to counter this threat, we used multiple research methods and presented BDD concepts to the participants. Furthermore, during interviews, we inquired the participants if they understood the BDD concepts and proposed process that we presented to them.

7. Conclusion

Large-scale product development involves a high level of complexity and requires interaction and collaboration among multiple stakeholders from various sub-products. Previous research has identified various aspects of large-scale product development that need improvement and further research. Behavior-driven development was introduced to facilitate product development by enhancing collaboration and by educating the practitioners about the business use-cases. In this study, we have evaluated the suitability of behavior-driven development (BDD) in the context of large-scale product development with the help of six workshop sessions, two sessions of BDD demonstrations for practitioners, by involving eighteen industrial practitioners from five large-scale organizations, and by interviewing seven experienced practitioners from industry. The study starts by identifying the supposed benefits and challenges related to the adaptation of BDD in a large-scale context. Later, a BDD inspired development process is proposed for large-scale product development. This process was evaluated in the industry and improved with the help of the feedback of practitioners. A summary of the three research questions is provided here.

RQ 1: What are the benefits practitioners associated with BDD in large-scale software projects? During multiple workshop sessions, practitioners were asked about the benefits they associated with a BDD based product development. Improved quality of software requirements is seen as a benefit as the template, Given, When, and Then, can help in describing requirements. Practitioners believed that the feature files containing system-level use-cases could help in developing a uniform understanding of requirements, which is perceived as a benefit of BDD. The reuse of artifacts in large-scale projects is also considered a benefit in a large-scale project. Furthermore, practitioners believed that test organization could benefit more if the organization moves to BDD. BDD reduces the chance of ambiguous requirements, and the stakeholders already agree with acceptance tests before the development start. These benefits can help in improving the overall quality of verification in a large-scale project.

RQ 2: What are the challenges that BDD leads to in large-scale software projects? Software practitioners were asked about the most significant challenges in the context of large-scale software development. They suggested that a lack of clear ownership in BDD and competence are critical challenges in the context of large-scale projects. The practitioner also believed that BDD is an expensive process to follow, as BDD requires detailed analysis and modeling of the scenarios, which can be costly in large-scale projects. They also listed that maintenance of BDD scenarios can be challenging because of the textual nature of scenarios. The difficulty in writing system-level test-case was also a challenge that practitioners mentioned. Lastly, the Versioning Control of BDD artifacts is also considered as a challenge in a large-scale context.

RQ 3: How can BDD be applied in large-scale software development projects? Based on the challenges listed in the previous research question, a BDD inspired development process is proposed in this study. The development process starts when a product manager suggests new behaviors for the product. These behaviors are then used by requirements engineers to search for existing similar behaviors to reuse the existing artifacts. In the next activity, the requirements engineer develops a system-level feature file that elaborates on the new behaviors in the form of features and scenarios. These scenarios are then developed into executable test-cases with the help of hooks (also called glue code). For the system-level verification, this executable system-level feature file provides the status of development completed for the new behaviors. This system-level feature file is then broken down into several sub-feature files belonging to each sub-product of large-scale products. Each sub-product then develops and verifies the sub-product code based on the sub-feature file. Once all the sub-feature files are verified, the feature moves to the verification unit. In the last stage, if the system-level behaviors pass, then features are considered as ready for release.

The process was evaluated in three steps, (i) by validating few activities of the process with the help of two industrial use-cases, (ii) by evaluating the complete process with the help of eighteen practitioners from five large-scale product development organizations, and (iii) by presenting and interviewing experienced software practitioners. Practitioners found this process useful, and they believe that the process can improve communication, quality, reuse, and documentation of large-scale production. However, few practitioners believed that the process could be costly to implement as it requires the involvement of multiple stakeholders, and their availability can be a bottleneck in the process. Practitioners suggested changes in the proposed process, such as a phase for architectural alignment and validating the format of feature files. Later, these changes were included in the final version of the proposed process. Practitioners were inquired about time to apply the process, and all of them believed that it might take more than a year to make this change in a large-scale development organization.

Future Work As mentioned by some of our investigation participants, introducing the proposed process in our organization may take more than a year. As a consequence, we could not perform the dynamic validation of our approach. Thus, we plan to run the dynamic validation as part of our future work. This application may help us in better understanding the characteristics of the process so that we can improve it further before releasing it for the software industry. Furthermore, we want to investigate the reusability of behaviors in BDD based methodology, providing guidelines to increase the reuse potential of these behaviors.

CRediT authorship contribution statement

Mohsin Irshad: Conceptualization, Methodology, Investigation, Formal analysis, Writing - original draft, Writing - review & editing. **Ricardo Britto:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Kai Petersen:** Writing - review & editing, Supervision.

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.

References

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empir. Softw. Eng. 16 (4), 487–513.

Ali, N.B., Petersen, K., Mäntylä, M.V., 2012. Testing highly complex system of systems: an industrial case study. In: Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 211–220.

Ali, N.B., Petersen, K., Schneider, K., 2016. Flow-assisted value stream mapping in the early phases of large-scale software development. J. Syst. Softw. 111, 213–227.

Bass, J.M., 2015. How product owner teams scale agile methods to large distributed enterprises. Empir. Softw. Eng. 20 (6), 1525–1557.

Begel, A., Nagappan, N., Poile, C., Layman, L., 2009. Coordination in large-scale software teams. In: Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering. IEEE Computer Society, pp. 1–7.

BehaveFramework, 2019. Behave. In https://behave.readthedocs.io/.

Binamungu, L.P., Embury, S.M., Konstantinou, N., 2018. Detecting duplicate examples in behaviour driven development specifications. In: 2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST). IEEE, pp. 6–10.

Bjarnason, E., Wnuk, K., Regnell, B., 2011. Requirements are slipping through the gaps—A case study on causes & effects of communication gaps in large-scale software development. In: 2011 IEEE 19th International Requirements Engineering Conference. IEEE, pp. 37–46.

Borg, R., Kropp, M., 2011. Automated acceptance test refactoring. In: Proceedings of the 4th Workshop on Refactoring Tools. ACM, pp. 15–21.

Britto, R., Smite, D., Damm, L.-O., 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). IEEE, pp. 1–11.

Bui-Thanh, T., Willcox, K., Ghattas, O., 2008. Model reduction for large-scale systems with high-dimensional parametric input space. SIAM J. Sci. Comput. 30 (6), 3270–3288.

Carrera, Á., Iglesias, C.A., Garijo, M., 2014. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. Inf. Syst. Front. 16 (2), 169–182.

Cisneros, L.A., Reis, C.I., Maximiano, M., Quiña, J.A., 2018. An experimental evaluation of itl, tdd and bdd. In: ICSEA 2018, the Thirteenth International Conference on Software Engineering Advances. ThinkMind, pp. 20–24.

Clarke, V., Braun, V., Hayfield, N., 2015. Thematic analysis. Qual. Psychol. Pract. Guide Res. Methods 222–248.

Cleland-Huang, J., Gotel, O.C., Huffman Hayes, J., Mäder, P., Zisman, A., 2014. Software traceability: trends and future directions. In: Future of Software Engineering Proceedings. pp. 55–69.

Cottam, J.A., Hursey, J., Lumsdaine, A., 2008. Representing unit test data for large scale software development. In: Proceedings of the 4th ACM Symposium on Software Visualization. pp. 57–66.

Coveros, 2019. Exploring glue code with cucumber-jvm. In https://www.coveros.com/exploring-glue-code-with-cucumber-jvm.

Dalal, S.R., McIntosh, A.A., 1994. When to stop testing for large software systems with changing code. IEEE Trans. Softw. Eng. 20 (4), 318–323.

De Almeida, E.C., Marynowski, J.E., Sunyé, G., Le Traon, Y., Valduriez, P., 2010. Efficient distributed test architectures for large-scale systems. In: IFIP International Conference on Testing Software and Systems. Springer, pp. 174–187.

de Carvalho, R.A., e Silva, F.L.d.C., Manhães, R.S., de Oliveira, G.L., 2013. Implementing behavior driven development in an open source erp. In: Enterprise Information Systems of the Future. Springer, pp. 242–249.

Diepenbeck, M., Soeken, M., Große, D., Drechsler, R., 2012. Behavior driven development for circuit design and verification. In: 2012 IEEE International High Level Design Validation and Test Workshop (HLDVT). IEEE, pp. 9–16.

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.

Dingsøyr, T., Fægri, T.E., Itkonen, J., 2013. What is large in large-scale? A taxonomy of scaling in agile software development. In: International Conference on Product-Focused Software Process Improvement. Springer, pp. 273–276.

Dingsøyr, T., Moe, N.B., 2013. Research challenges in large-scale agile software development. ACM SIGSOFT Softw. Eng. Notes 38 (5), 38–39.

Dingsøyr, T., Rolland, K., Moe, N.B., Seim, E.A., 2017. Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development. Procedia Comput. Sci. 121, 123–128.

Earley, S., 2014. The digital transformation: staying competitive. IT Prof. 16 (2), 58–60.

Egbreghts, A., 2017. A literature review of behavior driven development using grounded theory. In: 27th Twente Student Conference on IT. Available at: https://pdfs.semanticscholar.org/4f03/Ec0675d08cfd1ecdbaac3361a29d756ce656.pdf.

Feldt, R., Poulding, S., Clark, D., Yoo, S., 2016. Test set diameter: Quantifying the diversity of sets of test cases. In: 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE, pp. 223–233.

Gebhart, M., Giessler, P., Abeck, S., 2016. Challenges of the digital transformation in software engineering. In: ICSEA 2016. p. 149.

Glaser, B.G., Strauss, A.L., Strutzel, E., 1968. The discovery of grounded theory; strategies for qualitative research. Nurs. Res. 17 (4), 364.

Gohil, K., Alapati, N., Joglekar, S., 2011. Towards behavior driven operations (bdops). In: 3rd International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2011). IET, pp. 262–264.

Gorschek, T., Garre, P., Larsson, S., Wohlin, C., 2006. A model for technology transfer in practice. IEEE Softw. 23 (6), 88–95.

Guide, I., 2020. The interview guide to evaluate the process. In shorturl.at/hAHJS.

Guinan, P.J., Cooprider, J.G., Faraj, S., 1998. Enabling software development team performance during requirements definition: A behavioral versus technical approach. Inf. Syst. Res. 9 (2), 101–125.

Häser, F., Felderer, M., Breu, R., 2016. Is business domain language support beneficial for creating test case specifications: A controlled experiment. Inf. Softw. Technol. 79, 52–62.

Helgesson, D., Engström, E., Runeson, P., Bjarnason, E., 2019. Cognitive load drivers in large scale software development. In: Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE Press, pp. 91–94.

Irshad, M., Petersen, K., 2020. A method for supporting reuse of automated acceptance test. In Submission to a Conference But Available at: www.shorturl.at/aghyL.

Irshad, M., Petersen, K., Borstler, J., 2020. Reuse-potential: A measure to support refactoring in BDD. In Submission in a Journal But Available at: https://drive.google.com/file/d/1HWci8V-R5Ns0XpC1aP7G98LuLXmDLZH-/view?usp=sharing.

Jazayeri, M., 2002. On architectural stability and evolution. In: International Conference on Reliable Software Technologies. Springer, pp. 13–23.

Jørgensen, M., 2018. Do agile methods work for large software projects?. In: International Conference on Agile Software Development. Springer, pp. 179–190.

Kasauli, R., Liebel, G., Knauss, E., Gopakumar, S., Kanagwa, B., 2017. Requirements engineering challenges in large-scale agile system development. In: 2017 IEEE 25th International Requirements Engineering Conference (RE). IEEE, pp. 352–361.

Kettunen, P., Laanti, M., 2008. Combining agile software projects and large-scale organizational agility. Softw. Process Improv. Pract. 13 (2), 183–193.

Konrad, S., Gall, M., 2008. Requirements engineering in the development of large-scale systems. In: 2008 16th IEEE International Requirements Engineering Conference. IEEE, pp. 217–222.

Kumar, D., Mishra, K., 2016. The impacts of test automation on software's cost, quality and time to market. Procedia Comput. Sci. 79, 8–15.

Lazar, I., Motogna, S., Pârv, B., 2010. Behaviour-driven development of foundational uml components.. Electron. Notes Theor. Comput. Sci. 264 (1), 91–105.

Lethbridge, T.C., Sim, S.E., Singer, J., 2005. Studying software engineers: Data collection techniques for software field studies. Empir. Softw. Eng. 10 (3), 311–341.

Lethbridge, T.C., Singer, J., Forward, A., 2003. How software engineers use documentation: The state of the practice. IEEE Softw. 20 (6), 35–39.

Li, Y., Dong, T., Zhang, X., Song, Y.-d., Yuan, X., 2006. Large-scale software unit testing on the grid. In: GrC. pp. 596–599.

Linares-Vásquez, M., Moran, K., Poshyvanyk, D., 2017. Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp. 399–410.

Liu, Z., Mei, P., 2014. Automated testing for large-scale critical software systems. In: 2014 IEEE 5th International Conference on Software Engineering and Service Science. IEEE, pp. 200–203.

Lübke, D., van Lessen, T., 2016. Modeling test cases in bpmn for behavior-driven development. IEEE Softw. 33 (5), 15–21.

Mårtensson, T., Ståhl, D., Bosch, J., 2018. Enable more frequent integration of software in industry projects. J. Syst. Softw. 142, 223–236.

Melnik, G., Maurer, F., 2007. Multiple perspectives on executable acceptance test-driven development. In: International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 245–249.

Minhas, N.M., Petersen, K., Börstler, J., Wnuk, K., 2020. Regression testing for large-scale embedded software development–exploring the state of practice. Inf. Softw. Technol. 120, 106254.

Molléri, J.S., Petersen, K., Mendes, E., 2020. An empirically evaluated checklist for surveys in software engineering. Inf. Softw. Technol. 119, 106240.

North, D., 2006a. Introducing behaviour driven development. Better Softw. Mag..

North, D., 2006b. What's in a story?.

Obara, E., Kawasaki, T., Ookawa, Y., Maeda, N., 1996. Metrics and analyses in the test phase of large-scale software. In: Achieving Quality in Software. Springer, pp. 133–144.

Obbink, H., van Ommering, R., Wijnstra, J.G., America, P., 2002. Component oriented platform architecting for software intensive product families. In: Software Architectures and Component Technology. Springer, pp. 99–141.
Otaduy, I., Díaz, O., 2017. User acceptance testing for agile-developed web-based applications: Empowering customers through wikis and mind maps. J. Syst. Softw. 133, 212–229.
Perry, D.E., Siy, H.P., Votta, L.G., 2001. Parallel changes in large-scale software development: an observational case study. ACM Trans. Softw. Eng. Methodol. (TOSEM) 10 (3), 308–337.
Petersen, K., Wohlin, C., 2009. Context in industrial software engineering research. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 401–404.
Petersen, K., Wohlin, C., 2011. Measuring the flow in lean software development. Softw. - Pract. Exp. 41 (9), 975–996.
Pressman, R.S., 2005. Software Engineering: A Practitioner's Approach. Palgrave macmillan.
Questionnaire, 2019. Questionnaire. In Available at: www.shorturl.at/aghyL.
Rahman, M., Gao, J., 2015. A reusable automated acceptance testing architecture for microservices in behavior-driven development. In: 2015 IEEE Symposium on Service-Oriented System Engineering. IEEE, pp. 321–325.
RobotFramework, 2019. Robot framework. In https://robotframework.org.
Robson, C., McCartan, K., 2016. Real World Research. John Wiley & Sons.
Rocha, T., Borba, P., Santos, J.P., 2019. Using acceptance tests to predict files changed by programming tasks. J. Syst. Softw. 154, 176–195.
Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons.
Sabaliauskaite, G., Loconsole, A., Engström, E., Unterkalmsteiner, M., Regnell, B., Runeson, P., Gorschek, T., Feldt, R., 2010. Challenges in aligning requirements engineering and verification in a large-scale industrial context. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 128–142.
Scacchi, W., 1989. Engineering large-scale software systems: an organizational knowledge base approach. In: Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage. IEEE, pp. 232–235.
Script, J., 2020. The script to calculate ncd. In shorturl.at/agjIO.
Smite, D., Moe, N.B., Levinta, G., Floryan, M., 2019. Spotify guilds: How to succeed with knowledge sharing in large-scale agile organizations. IEEE Softw. 36 (2), 51–57.
Soeken, M., Wille, R., Drechsler, R., 2012. Assisted behavior driven development using natural language processing. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer, pp. 269–287.
Solis, C., Wang, X., 2011. A study of the characteristics of behaviour driven development. In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, pp. 383–387.
Stray, V., Moe, N.B., Aasheim, A., 2019. Dependency management in large-scale agile: a case study of DevOps teams. In: Proceedings of the 52nd Hawaii International Conference on System Sciences.
Usman, M., Britto, R., Damm, L.-O., Börstler, J., 2018. Effort estimation in large-scale software development: An industrial case study. Inf. Softw. Technol. 99, 21–40.
Vierhauser, M., Rabiser, R., Grünbacher, P., 2014. A case study on testing, commissioning, and operation of very-large-scale software systems. In: Companion Proceedings of the 36th International Conference on Software Engineering. pp. 125–134.
Zaïane, O.R., Antonie, M.-L., 2002. Classifying text documents by associating terms with text categories. In: Proceedings of the 13th Australasian Database Conference-Volume 5. pp. 215–222.

Mohsin Irshad (mohsin.irshad@bth.se): Mohsin Irshad is a software engineer at Ericsson, Sweden, and a Ph.D. student at Blekinge Institute of Technology (BTH), Sweden. Mohsin has a proven track record in the software industry with 10+ years of experience working with different telecommunication vendors. In 2018, Mohsin earned Licentiate in Software Engineering from the Blekinge Institute of Technology (BTH), Sweden. His research interests are in Software development, Software Testing, Machine learning, and Evidence-Based Software Engineering.

Affiliation: Ericsson AB, Karlskrona & Blekinge Institute of Technology, Software Engineering Research Lab, Blekinge Institute of Technology, Valhallavgen 1, 371 41 Karlskrona, Sweden

Ricardo Britto (ricardo.britto@bth.se): 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.

Affiliation: Ericsson AB, Karlskrona & Blekinge Institute of Technology, Software Engineering Research Lab, Blekinge Institute of Technology, Valhallavägen 1, 371 41 Karlskrona, Sweden

Kai Petersen (kai.petersen@bth.se): Kai Petersen is a professor of software engineering at Blekinge Institute of Technology (BTH), Sweden and University of Applied Sciences Flensburg, Germany. He received his Ph.D. from BTH in 2010. His research interests are Agile Software Development, Software Testing, Evidence-Based Software Engineering and Software Measurement. His research has been conducted in close collaboration with companies and with an empirical focus.

Affiliation: Blekinge Institute of Technology, Software Engineering Research Lab, Blekinge Institute of Technology, Valhallavägen 1, 371 41 Karlskrona, Sweden