Researching Cooperation and Communication in Continuous Software Engineering

Yvonne Dittrich
Dep. of Computer Science
IT University of Copenhagen
Copenhagen, Denmark
ydi@itu.dk

Jacob Nørbjerg
Dep. of Digitalization
Copenhagen Business School
Copenhagen, Denmark
jno.digi@cbs.dk

Paolo Tell
Dep. of Computer Science
IT University of Copenhagen
Copenhagen, Denmark
pate@itu.dk

Lars Bendix
Dep. of Computer Science
Lund University
Lund, Sweden
bendix@cs.lth.se

ABSTRACT

Continuous Software Engineering (CSE)—continuous development and deployment of software—and DevOps—the close cooperation or integration of operations and software development—is about to change how software is developed. Together with the tighter integration of development and operations also with usage this will change coordination and collaboration both between IT professionals and between developers and users. In this short paper, we discuss the CHASE dimension of three core research themes that begin to crystallize in literature. This position paper is intended as a 'call to arms' for the CHASE community to study CSE.

CCS CONCEPTS

• **Software and its engineering → Agile software development**; **Programming teams**;

KEYWORDS

Continuous Software Engineering, DevOps

ACM Reference Format:
Yvonne Dittrich, Jacob Nørbjerg, Paolo Tell, and Lars Bendix. 2018. Researching Cooperation and Communication in Continuous Software Engineering. In *CHASE'18: CHASE'18:IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3195836.3195856

1 INTRODUCTION

Continuous Software Engineering (CSE) evolved from the need to meet quality-of-service requirements in large online service providers like Google, Amazon, and Facebook. These companies developed tools and practices to correct errors and update functionality in on-line applications without downtime, resulting in the automation of build, test, and deployment, and the replacement of scheduled release cycles by evolving requirements and continuous development and release of new software versions. The advantages of these practices are: new features can be quickly deployed and tested; shorter time to meet changing business needs; user feedback can be collected and applied on an ongoing basis; different features and interfaces can be compared in production; higher developer productivity and satisfaction; and, last but not least, the dynamic adaptation to demands attracts new customers.

Software companies in other sectors (e.g. financial institutions and embedded systems) are also beginning to explore CSE. They may be forced by legislation, customer requirements, and international competition to, e.g., quickly implement changes to compliance rules in the financial sector; ward of competitors in the online payment sector through rapid implementation of new features; and to survive as a small software company in a highly competitive environment through rapid response to changing customer demands while maintaining high quality. The changes to the development practice will change the cooperation between software engineers and users, respectively customers as well.

It is, therefore, now generally recognized that CSE will become a game changer for the software industry [29]. And this will impact how software development as a collaborative practice will unfold. CHASE research has an important contribution here: As we argue below, the core research themes discussed in the context of CSE all point to CHASE topics as central for handling the outlined challenges.

This short paper is based on an initial literature study complemented by several student projects and a series of interviews that have not yet been fully analyzed. The paper, therefore, takes the form of a position paper that argues for CHASE research on CSE. The insights into industrial practice gained from the student projects and the interviews complement the findings from the related work and especially add a social and collaborative dimension. The three themes below emerge from core challenges reported in the interviews with industrial practitioners.

CHASE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5725-8/18/05…$15.00
https://doi.org/10.1145/3195836.3195856

The following section presents a short literature review. Section 3 then discusses the three research themes and their CHASE dimensions. Section 4 presents our research approach before the conclusion of the paper.

2 CONTINUOUS SOFTWARE ENGINEERING

"Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)" [19]. In spite of the challenges facing companies moving towards CSE, there is a scarcity of documented and validated guidelines for adopting CSE practices, tools, and processes across sectors and within companies of different sizes; c.f.; [6, 11, 19].

Existing case studies and guidebooks promote solutions to partial problems without discussing the implications of their application: For instance, self-contained microservices are suggested as an architectural pattern for CSE [2], but such a design requires extra measures to maintain data consistency and support analysis across services. Even the best studies, such as the account of Microsoft's change of the Team Foundation Server™ to the cloud-based Visual Studio™ [14], discuss a specific journey to CSE, but do not provide orientation for the transition of other companies or within other domains.

Shahin et al. [24] list 13 CSE adoption practices that have been reported in the literature, but do not present advice on how to select among them in different circumstances. Neely and Stolt [19] offer specific advice based on the experiences from their organization, but the advice is based on one case only. The Gartner group has published a roadmap for CSE [29], but its empirical foundation is unclear.

The most comprehensive research-based roadmap, the Stairway to Heaven framework by Olsson and Bosch [22], specifies 5 stages on the journey towards CSE: traditional development, R&D organization all agile, continuous integration, continuous deployment, and R&D as an Innovation system. This final step is similar to the continuous loop from business, over development, deployment and use, and back to business described by Fitzgerald and Stol [11].

The Stairway to Heaven framework is based on observational studies in the embedded software and telecommunications sectors, and even though it, and the other studies and experience reports in [25] comprise a useful milestone in CSE research, it has also been criticized for assuming an organizational view, and ignoring the lower level managerial, process, and personal-cognitive issues involved in changing towards CSE [27]. At the level of software processes and management, for example, Dennehy and Conboy [8] discuss the frictions created when plan and control responsibilities move from the software team to managers external to the team, and decisions become based on measurements instead of the knowledge and experience of the team. The framework also fails to integrate the relationship between the customer and software organizations at the strategic and budgeting levels [1, 11].

Especially reading and listening to various experience reports, it becomes very clear that CSE will change how we cooperate and coordinate software engineering. As there is no common release cycle anymore, different software engineering activities as well as different teams will develop their specific rhythm. Crosscutting tasks require new coordination mechanisms [23]. Coordination both across different activities and across different modules of a software system will have to be rethought.

Below, we further discuss three research themes that are recurring both in research literature and our discussions with practitioners.

3 THREE RESEARCH THEMES

Whereas related work focuses on continuous processes across business, development and operations (e.g. [11]) and proposes maturity models (e.g. [22]), the industrial practitioners focus on underpinning practices that indicate the core challenges of continuous development: design and architecture to support CSE, quality and test automation, and changing processes and management within and beyond the software organization. These themes are discussed below and in relation to specific related work as well as relevant student projects (MSc theses) and interviews.

3.1 Tooling and Architecting

The technical design of software, and the integration, test and deployment infrastructures must evolve to support CSE processes. The necessity to treat IT infrastructures as code adds new requirements to test and release automation tools. Massive incremental change adds requirements to the technical design of software. Such design, in turn, requires new strategies for code and dependency management. Furthermore, many organizations struggle with redesigning their legacy systems architecture to fit the demands of CSE.

Research on architectural design to support CSE is still in an immature stage. There is a fairly clear understanding of the problem—managing the dependencies between code and (virtual) integration, test, and deployment infrastructures [15]—and there is a growing realization that a microservices architecture might be one possible solution [2]. However, current architectural research is only helpful for the few companies that can start from scratch. The question of how companies can migrate from an established monolithic to a CSE-friendly architecture is still unresolved. Nor is there any help for companies that for some reason cannot migrate.

An example for unresolved issues is cross-service data integration and management. CSE based on a microservice architecture assumes that the data model and the services are partitioned in similar ways. In data intensive businesses, the common data model is one of the core assets and dependencies are often through the data model [26]. Partitioning this data model might result in replication of data, which again provides additional challenges. Cross service data analysis requires a whole independent infrastructure that again needs to be carefully designed so that it can evolve together with the services and their data. All these dependencies have to be managed by software engineers and architects [7].

CSE requires a new way of handling code ownership and coordination of the implementation of cross-cutting features beyond the decision on a specific architecture. Continuous integration and testing are a prerequisite to be able to discover conflicts and regressions as soon as possible. Some of our interview partners work internally with open source like models of code ownership. If changes to several services are necessary to implement a feature, multiple teams

might be involved, or pull requests for the change of a service owned by another team are submitted.

Thus, continuous software engineering will require new patterns of coordination and cooperation in software development.

3.2 Continuous Quality Assurance

Software quality control is often stated as one of the key enablers of CSE [11]. CSE requires developers to continuously test the software under development. The implications for the cooperation between software developers and testers, and their practices, however, are seldom discussed.

The core measures here are test automation and live site monitoring. For CSE, unit test automation needs to be extended to integration and acceptance test levels. Frameworks and infrastructures to support such automatisation have been developed, e.g., in the context of Behavior Driven Development [5, 21]. An initial study showed that rather than the deployment of tools and methods, the changes of work practices and the cooperation between different roles are the core challenges [16]. Testers need to work closer together with software engineers. The development and maintenance of the test suite needs to become part of the continuous evolution of the code. The introduction of code scanning tools in the continuous integration environment likewise results in challenges for the individual developer and requires changes to the development practice: coding standards need to be agreed on in advance; the team needs to agree on what is considered good coding practices; the code architecture and architectural design principles need to be explicated [3].

CSE allows to close the feedback loop between software development and use through instrumentation and telemetry. The challenge here is to use telemetry so the data helps understanding customer needs. Research on site monitoring [28] focuses on tools and techniques, and often fails to explore the repercussions for the software process. The experience report of Microsoft's Visual Studio indicates that close contact between the development team and pilot users is as important as the careful design of instrumentation and experiments.

In regulated domains, like finance and health care, the need to re-structure the cooperation for quality control extends into the business departments: In Denmark, software development and deployment in these domains require a manual accept by a business representative, but can a business representative check core test cases several times a day? This leads to the next theme discussed below.

3.3 CSE Processes and Interaction with Business and Users

Most CSE literature focuses on activities within software development (cf. [24]), while implications for the cooperation with customers and users are mostly ignored or discussed at a superficial level only. CSE requires a radical change to how software development is coordinated and managed, however. This is currently only rudimentary understood.

User-Centered Design (UCD) in the context of agile development has been discussed since the dawn of agile development [4]. The continuous stakeholder involvement, interweaving of UCD and agile development, and the focus on artefact-mediated cooperation between different professionals, are all indicated by the systematic literature study in [4] as examples of principles that can be a starting point to combine UCD and continuous development. However, as the development does not follow common release cycles anymore, and UCD activities according to our experiences have a different cadence than programming and development, the integration of UCD and software engineering in continuous software engineering can be expected to take a different shape.

In one company, developers and managers, implemented and shadow deployed more innovative developments through an independent pipeline parallel to the production environment, and merged the innovations back to the main source code once they had been proven and accepted.

Further, what changes are needed on the boundary between development and user organizations? Will users and customers adapt to short development and release cycles?

In line with [11], we view CSE as a continuous flow from business strategy and decision making over development, to operations and use, and finally returning user feedback and new ideas back to the business. From this perspective, CSE means the continuous and incremental change to an organization and the IT systems that it produces and/or uses. This however raises yet another question: Will software developers and user organizations be able to combine the incremental *improvement* to processes of CSE with creativity and radical innovation [11]?

Very little research discusses how to transform not only the software but also user/customer organizations in order to reach the stage of effective and efficient CSE.

4 HOW TO RESEARCH THE CHASE SIDE OF CSE

Industrial practice is asking for research-based actionable guidelines and advice in order to harvest the advantage of CSE [6, 13]. As we argued above, collaboration and human aspects are central to the agreed-on research challenges. Researching CSE now will allow CHASE research to influence the still emerging paradigm and both develop the CHASE body of knowledge and become influential in supporting the development of techniques, tools and processes supporting the collaborative and human aspects of CSE.

Researching companies in the transition from conventional and agile development to CSE will allow CHASE researchers to explore the differences between CSE and previous approaches to coordination and collaboration and will support the development of theoretical explanations.

A relevant research approach is Action Research. Action Research combines structured, trustworthy empirical research with method, technique and tool development (see [10, 17, 18, 20]). Cooperative Method Development (CMD) [10], for example, proposes a series of action research cycles, each with 3 phases: (1) understand current practices and problems; (2) deliberate change; and (3) observe and evaluate improvements.

In order to support the adaptation of the research results to heterogeneous contexts and sectors, and support Software as a Service provisioning as well as complex infrastructures, we do not see a new mega method as the way to move forward. We rather propose

to formulate methods as practice patterns [9]. Applying the concept of design patterns [12] to Software Engineering methods, relates the proposed solution not only to the context and the problem, but allows to detail forces that influence the adaptation of the solution, as well as consequences and implications for other patterns. Such patterns can support the transition to CSE as they provide the necessary information to pick and experiment based on the additional information the method patterns provide.

Above, we argued that the new processes and collaboration structures are interacting with software architecture, tools, and QA. However, the changes will affect the interaction with business stakeholders and users as well. Research and research results in form of method patterns need to cover this cooperation too.

Action research further provides the opportunity to establish and support continuous improvement, innovation and experimentation [11] at the same time as researching both the learning and the CSE practices.

5 CONCLUSIONS

Continuous software engineering will change how we develop software. It therefore will also change how developers collaborate with each other and with other stakeholders in the software development and deployment process. Communication, coordination and cooperation will take place differently from what we have seen so far. CHASE research, therefore, has a unique role to play here.

ACKNOWLEDGMENT

The authors would like to thank all the practitioners that contributed time and insights to this research.

REFERENCES

[1] Ritu Agarwal and Amrit Tiwana. 2015. Editorial-Evolvable Systems: Through the Looking Glass of IS. *Info. Sys. Research* 26, 3 (Sept. 2015), 473–479. https://doi.org/10.1287/isre.2015.0595

[2] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional.

[3] Brynjolfur Bjarnason. 2016. Managing Technical Depth. Master Thesis. IT University of Copenhagen. (2016).

[4] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. 2015. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology* 61 (2015), 163–181.

[5] J. Carter and W. B. Gardner. 2016. BHive: Towards Behaviour-Driven Development Supported by B-Method. In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI).* 249–256. https://doi.org/10.1109/IRI.2016.39

[6] Lianping Chen. 2015. Continuous delivery: Huge benefits, but challenges too. *IEEE Software* 32, 2 (2015), 50–54.

[7] Cleidson RB de Souza and David F Redmiles. 2009. On the roles of APIs in the coordination of collaborative software development. *Computer Supported Cooperative Work (CSCW)* 18, 5-6 (2009), 445.

[8] Denis Dennehy and Kieran Conboy. 2017. Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software* 133 (2017), 160–173.

[9] Yvonne Dittrich. 2016. What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology* 70 (2016), 220–231.

[10] Yvonne Dittrich, Kari Rönkkö, Jeanette Eriksson, Christina Hansson, and Olle Lindeberg. 2008. Cooperative method development. *Empirical Software Engineering* 13, 3 (2008), 231–260.

[11] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming.* Springer, 406–431.

[13] Peggy Gregory, Leonor Barroca, Helen Sharp, Advait Deshpande, and Katie Taylor. 2016. The challenges that challenge: Engaging with agile practitionersâĂŹ concerns. *Information and Software Technology* 77 (2016), 92–104.

[14] Sam Guckenheimer. 2016. *Our journey to cloud cadence lessons learned at Microsoft developer division.* Technical Report. https://www.microsoft.com/en-us/download/details.aspx?id=4692

[15] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Pearson Education.

[16] Mark Klittgard and Rasmus Rosted. 2017. Introducing new Procedures in a Busy Environment. A Cooperative Method Development Study on Continuous Software Engineering in a Mediom Sized Company. Master Thesis. IT University of Copenhagen. (2017).

[17] Lars Mathiassen. 2002. Collaborative practice research. *Information Technology & People* 15, 4 (2002), 321–345.

[18] Lars Mathiassen, Peter Axel Nielsen, and Jan Pries-Heje. 2002. *Learning SPI in practice.* Addison-Wesley, Boston.

[19] Steve Neely and Steve Stolt. 2013. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *Agile Conference (AGILE), 2013.* IEEE, 121–128.

[20] Peter Axel Nielsen and Karlheinz Kautz. 2008. *Software process & knowledge: Beyond conventional software process improvement.* Software Innovation Publisher.

[21] Dan North. 2006. Behavior Modification: The evolution of behavior-driven development. *Better Software* 8, 3 (2006).

[22] Helena Holmström Olsson and Jan Bosch. 2014. Climbing the "Stairway to Heaven": evolving from agile development to continuous deployment of software. In *Continuous software engineering.* Springer, 15–27.

[23] Kjeld Schmidt and Carla Simonee. 1996. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work (CSCW)* 5, 2-3 (1996), 155–200.

[24] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access* 5 (2017), 3909–3943.

[25] Daniel Ståhl and Jan Bosch. 2014. Continuous integration flows. In *Continuous software engineering.* Springer, 107–115.

[26] Kenn Thisted. 2014. Rethinking Release – A Case Study. Master Thesis. IT University of Copenhagen. (2014).

[27] Andre van Hoorn, Pooyan Jamshidi, Philipp Leitner, and Ingo Weber. 2017. Report from GI-Dagstuhl Seminar 16394: Software Performance Engineering in the DevOps World. *arXiv preprint arXiv:1709.08951* (2017).

[28] André van Hoorn, Matthias Rohr, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey, and Dennis Kieselhorst. 2009. Continuous monitoring of software services: Design and application of the Kieker framework. (2009).

[29] Nathan Wilson. 2017. Modernizing Application Development Primer for 2017. Gartner inc. (2017).