Information and Software Technology 86 (2017) 87–100

Contents lists available at ScienceDirect

Information and Software Technology

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

Systematic literature review on the impacts of agile release engineering practices

Teemu Karvonen*, Woubshet Behutiye, Markku Oivo, Pasi Kuvaja

University of Oulu, Pentti Kaiteran Katu 1, 90014, Finland

ARTICLE INFO

Article history:
Received 23 May 2016
Revised 12 January 2017
Accepted 24 January 2017
Available online 25 January 2017

Keywords:
Release engineering
Agile
Continuous integration
Rapid release
Continuous delivery
Continuous deployment

ABSTRACT

Context: Agile release engineering (ARE) practices are designed to deliver software faster and cheaper to end users; hence, claims of such impacts should be validated by rigorous and relevant empirical studies.
Objective: The study objective was to analyze both direct and indirect impacts of ARE practices as well as to determine how they have been empirically studied.
Method: The study applied the systematic literature review research method. ARE practices were identified in empirical studies by searching articles for "rapid release," "continuous integration," "continuous delivery," and "continuous deployment." We systematically analyzed 619 articles and selected 71 primary studies for deeper investigation. The impacts of ARE practices were analyzed from three viewpoints: impacts associated with adoption of the practice, prevalence of the practice, and success of software development.
Results: The results indicated that ARE practices can create shorter lead times and better communication within and between development teams. However, challenges and drawbacks were also found in change management, software quality assurance, and stakeholder acceptance. The analysis revealed that 33 out of 71 primary studies were casual experience reports that had neither an explicit research method nor a data collection approach specified, and 23 out of 38 empirical studies applied qualitative methods, such as interviews, among practitioners. Additionally, 12 studies applied quantitative methods, such as mining of software repositories. Only three empirical studies combined these research approaches.
Conclusion: ARE practices can contribute to improved efficiency of the development process. Moreover, release stakeholders can develop a better understanding of the software project's status. Future empirical studies should consider the comprehensive reporting of the context and how the practice is implemented instead of merely referring to usage of the practice. In addition, different stakeholder points of view, such as customer perceptions regarding ARE practices, still clearly require further research.

1. Introduction

In software development, fast, incremental deliveries involve lightweight, efficient practices for continuous release planning [1] and release engineering [2,3]. This paper focuses on engineering by synthesizing empirical studies for *agile release engineering* (ARE) practices. To the best of our knowledge, the concept of ARE has not been used before in other scientific papers. We use it in this paper to map the research topic and to incorporate the investigated release engineering practices that are involved in a *"release engineering pipeline"* [3]. Hence, by ARE practices, we mean contemporary software *integration, testing, deployment* and *release* practices that are often applied in modern release engineering [3]. Many of these practices originate from agile software development methodologies such as extreme programming (XP) [4]. From the research point of view, ARE practices derive from the theories of *agile and lean software development* [5], *release engineering* [2,3,6] and *continuous software engineering* [7,8] research disciplines. According to Adams et al. [6], release engineering "deals with all activities in between regular development and delivery of a software product to the end user, i.e., integration, build, test execution, packaging and delivery of software." Continuous software engineering is an emerging subtopic in software engineering (SE) that is focused on continuous experimentation, innovation and the elimination of discontinuities within and between the developmental, operational and business strategy functions. Fitzgerald and Stol [8] associate concepts in continuous software engineering with concepts of classic "lean thinking" [9] such as "value and waste,"

* Corresponding author.
E-mail addresses: teemu.3.karvonen@oulu.fi (T. Karvonen), woubshet.behutiye@oulu.fi (W. Behutiye), markku.oivo@oulu.fi (M. Oivo), pasi.kuvaja@oulu.fi (P. Kuvaja).

http://dx.doi.org/10.1016/j.infsof.2017.01.009
0950-5849/

"flow and batch size," "autonomation and building-in quality" and "Kaizen and continuous improvement."

Modern ARE practices are supposed to aid in delivering software faster and cheaper to end users; hence, claims of such impacts should be validated by rigorous and relevant empirical studies. In this systematic review, the objective is to understand the direct and indirect impacts of ARE practices. By investigating the direct and indirect impacts we emphasize the notion that "impacts may be desired already according to the explicit method rationale(s), or they may be unexpected, sometimes even unwanted[10]." In addition, we sought to evaluate primary studies to understand how the impacts of ARE practices have been investigated in empirical studies. The research questions are:

RQ1: What are the direct and indirect impacts of ARE practices?
We break down the main question into three sub-questions as follows:
RQ1a: What are the impacts associated with adoption of ARE practices?
RQ1b: What is the prevalence of ARE practices?
RQ1c: What are the impacts of ARE practices on the success of SW development?

In addition, we define a second main research question to understand how these impacts have been investigated:

RQ2: How have ARE practices been investigated in empirical studies?

In this study, we systematically searched and analyzed empirical studies investigating 1) *continuous integration* (CI) [11,12], 2) *continuous delivery* (CD) [12], 3) *rapid release* (RR) [13] and 4) *continuous deployment* (CD2) [11,12]. We analyzed and clustered studies by topic and research approach, and outlined a checklist for analyzing software development capabilities for CD2 in the context of software-intensive products. We applied a systematic literature review (SLR) [14] method that allowed us to critically compare, evaluate and synthesize the primary studies. Our main selection criterion for the primary studies was that they were conducted in real software development contexts. Literature reviews, mapping studies, opinion papers and small-scale experiments with students are not included in our analysis. To the best of our knowledge, a systematic synthesis focusing on the impacts of ARE practices has not previously been undertaken, although some of the practices have been synthesized either separately or from a different research question point of view, as we explain in more detail in the following section. With this systematic review, we aim to provide a reliable overview of the current state of existing empirical studies for ARE practices that may help in terms of scoping and planning future studies. Our study also helps practitioners to better understand the impacts and capabilities associated with ARE practices. Finally, this paper aims to contribute to the theorizing on software development practices [10] for ARE. The concepts used in this paper (i.e., *learning, practice, development context, rationale, impact* and *theory*) conform to definitions used for the Coat Hanger model [10].

2. Background

ARE practices aim to support the agile principle of "early and continuous delivery of valuable software [15]." Early and continuous deliveries allow mechanisms for fast feedback and transparency of the development process, allowing stakeholders to continuously review and evaluate the state of the system under development and, if needed, to make adjustments to the priority and content requirements accordingly. CI practice originated from the agile XP methodology. Beck and Andres [4] summarizes CI as follows: "New code is integrated with the current system after no more than a few hours. When integrating, the whole system is built from scratch and all tests must pass or the changes are discarded." CI is often characterized by development conventions and tools for the automation of the build and test activities [11]; XP, however, goes beyond tools and emphasizes *values* and *principles* [4] that *rationalize* [10] the usage of CI practice. Development team members must prioritize development activities to "commit changes frequently" and "fix broken builds immediately [4]." Consequently, "all tests and inspections must pass" (i.e., with zero tolerance for regression and failed test cases) [4].

It is safe to assume that the impacts of CI practice depend on the actual implementation of the practice as well as the stage of assimilating the practice the organization is in. In their literature review, Eck et al. [16] investigated CI practice from the assimilation point of view. They identified 14 distinct organizational implications of CI in three assimilation stages:

1. Acceptance: devising an assimilation path, overcoming the initial learning phase, dealing with test failures right away and introducing CI for complex systems.
2. Routinization: institutionalizing CI, clarifying the division of labor, CI and distributed development, mastering test-driven development and providing CI at project start.
3. Infusion: CI assimilation metrics, devising a branching strategy, decreasing test result latency, fostering customer involvement in testing and extending CI beyond the source code.

Another related literature review by Ståhl and Bosch [17] focused on technical aspects of the implementation of CI practice. Moreover, they analyzed variations in the interpretation and implementation of CI practice. They concluded that CI practice interpretations and implementations are different from case to case. For example, integration and test flow frequency vary in different contexts. Meanwhile, Päivärinta and Smolander [10] state that predefined practices or methods in theory need to be distinguished from the contextual practice descriptions of what actually happens. Subsequently identified CI impacts, as well as other ARE practices in our systematic review, may not apply to practice in theory but rather to different variants (context-specific embodiments) of the practice, which may evolve dynamically over time. Ståhl and Bosch [17] also state that "in order to make a meaningful comparison of software development projects, simply stating that they use continuous integration is insufficient information as we instead need to ask ourselves what kind of continuous integration is used." To better understand CI practice variations, Ståhl and Bosch introduced a descriptive model that allows for the visualization of CI flow for a better understanding of context-specific variation points. Nevertheless, understanding technical implementations of CI practice alone may still not provide enough information to understand the causalities between CI practice and its impacts, as many other variables may be related to the adoption of its values and principles that may also either increase or diminish the impacts.

Within CI practice, multiple levels of integration may exist and changes may also be deployed in different kinds of environments. For example, the deployment environment could be a purely experimental test environment, a production-like environment or the actual production environment. CD practice is often considered to extend CI practice. Humble and Farley [12] introduce CD as a practice to automate the software delivery process for production purposes. CD promotes the idea of delivering software "at will," i.e., the delivery can occur at any point in time with very little manual labor required. CD is often used interchangeably or as a synonym for CD2 practice; however, there are also accounts regarding how these practices differ. According to Fitzgerald and Stol [8], "continuous delivery is a prerequisite for continuous deployment, but