Information and Software Technology 000 (2016) 1–18

Contents lists available at ScienceDirect

Information and Software Technology

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

Cinders: The continuous integration and delivery architecture framework

Daniel Ståhl[a,*], Jan Bosch[b]

[a] Ericsson AB, Datalinjen 3, 581 12 Linköping, Sweden
[b] Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 11 May 2016
Revised 18 November 2016
Accepted 22 November 2016
Available online xxx

Keywords:
Cinders
Software integration
Software testing
Continuous integration
Continuous delivery
Architecture framework

ABSTRACT

Context: The popular agile practices of continuous integration and delivery have become an essential part of the software development process in many companies, yet effective methods and tools to support design, description and communication of continuous integration and delivery systems are lacking.

Objective: The work reported on in this paper addresses that lack by presenting Cinders — an architecture framework designed specifically to meet the needs of such systems, influenced both by prominent enterprise and software architecture frameworks as well as experiences from continuous integration and delivery modeling in industry.

Method: The state of the art for systematic design and description of continuous integration and delivery systems is established through review of literature, whereupon a proposal for an architecture framework addressing requirements derived from continuous integration and delivery modeling experiences is proposed. This framework is subsequently evaluated through interviews and workshops with engineers in varying roles in three independent companies.

Results: Cinders, an architecture framework designed specifically for the purpose of describing continuous integration and delivery systems is proposed and confirmed to constitute an improvement over previous methods. This work presents software professionals with a demonstrably effective method for describing their continuous integration and delivery systems from multiple points of view and supporting multiple use-cases, including system design, communication and documentation.

Conclusion: It is concluded that an architecture framework for the continuous integration and delivery domain has value; at the same time potential for further improvement is identified, particularly in the area of tool support for data collection as well as for manual modeling.

1. Introduction

This paper addresses the problem of constructing systems for rapidly and frequently transforming source code changes into verified and deliverable software product revisions with known content, known functionality and known quality — through software integration and test — in a controlled and methodical way.

Historically, the problem of transforming lines of source code into functioning, verified products running in their target environment could be regarded as a question of enterprise architecture: of organizational responsibilities and manual processes. With the advent and growth of continuous integration [2,8] and delivery [9,14], however, and the automation this brings, this is increasingly becoming a domain of software engineering: we see ever more sophisticated software systems being constructed, with the purpose of compiling, integrating, testing, delivering and deploying other software.

While such systems are generally perceived as adding value and increasing the efficiency of the development project, we have found in previous work that the exact nature of these benefits is highly uncertain and varies from case to case [35]. Furthermore, even though there are numerous popular tools that do much of the heavy lifting in these integration systems, they only address isolated parts of a very large problem domain. In all our industry case studies [35,36,38,39] we have never found a complete off-the-shelf solution for continuous integration. Rather, the integration systems we find often use similar tools, but configured differently, put to different purposes and integrated with one another in varying constellations. Not surprisingly, a review of literature reveals that reported continuous integration systems display a high degree of variance [37]. In other words, as a rule, continuous inte-

* Corresponding author.
E-mail addresses: daniel.stahl@ericsson.com (D. Ståhl), jan@janbosch.com (J. Bosch).

http://dx.doi.org/10.1016/j.infsof.2016.11.006
0950-5849/

gration and delivery systems are highly customized and purpose-built software products in their own right.

Similarly to the variance in system design, there is little consensus on the exact definition of continuous integration and delivery, particularly as opposed to related terms such as continuous testing, continuous release or continuous deployment. For the purposes of this paper, we use the term *continuous integration and delivery system* to mean any system of automated activities performed in order to transform source code into working and potentially shippable and deployable products with known quality, content and functionality, i.e. including compilation, linking, packaging, testing, profiling, documentation generation and much more, serving to ensure that "the software can be released to production at any time" [9]. In this paper we propose and evaluate a structured and systematic approach to the design and description of such systems, in order to help software professionals become more effective and efficient in a critical domain where significant resources are currently being invested.

The key contribution of this paper is that it provides an architectural framework for the design and description of continuous integration and delivery systems, thereby addressing the lack of systematic engineering approaches to solving a critical problem in the software development industry.

The rest of the paper is structured as follows. The problem statement is presented in the following section, discussing in greater depth the relevancy of systematic approaches to continuous integration delivery system design and description. Following this, Section 3 provides a background to the work reported from in this paper by discussing related tools and methods as well as previous work on continuous integration modeling. Section 4 presents the employed research method. Then the results are presented in Section 5 and evaluated in Section 6. Threats to validity are discussed in Section 7 and the paper is then concluded in Section 8.

2. Problem statement

It is easy to find introductions and tutorials to continuous integration — apart from a number of books, a web search for ''continuous integration tutorial'' returns well over a thousand hits [10]. While a small development project may require little work in order to set up a simplistic solution, designing a reliable continuous integration and delivery system for a medium to large scale project — particularly one satisfying the high levels of traceability required in large segments of industry [3,7,26,27] — is not a small undertaking, however. In addition, all such large scale cases studied by us in previous work [35,36,38,39], while incorporating many of the same open source or commercial tools, are ultimately in-house developed bespoke systems.

The fact that continuous integration is non-trivial to scale has been recognized in literature for some time [29,31]. It is also known that large numbers of different tools, such as build tools, version control systems and test automation frameworks, are involved in continuous integration and delivery [12,19,44] along with numerous stakeholder roles [22]. We have ourselves noted the importance of considering the continuous integration and delivery system's *capacity*, in relation to the needs dictated by one's project size, and discussed strategies for handling that [36]. We believe that the problems arising from this are further compacted by the fact that the companies — despite long experience of developing large, highly systematized products on a multinational market — tend to not approach this area with the same rigorous engineering practices as they do their normal product development. Arguably because of the emergent nature of the field, there appears to be a lowering of standards when developing the systems that ultimately produce the products, compared to the development of the products themselves. This notion is supported by statements made by industry professionals interviewed by us in previous work [38], e.g. an integration project manager of a very large software product relating how they simply don't approach the problem in a systematic way and don't document what exactly it is they want to achieve. In the same vein, a test manager in another company previously studied by us explained how he constantly struggles with overlapping and competing in-house tool development, leading to enormous waste of resources — a situation unthinkable in the development of the regular product itself — simply because of insufficient coordination of development teams and little to no systemization of, in his words, "the continuous integration and delivery product".

Further evidence of the vast engineering problems involved in building large, effective and reliable continuous integration and delivery systems is provided by the effort required. While it is surprisingly difficult to find studies on capital and operating expenditure of such implementations, one study [24] estimates that a small team of developers spent approximately 7% of their effort over a hundred day period on "CI process overhead", although this was outweighed by the assumed cost of *not* having implemented continuous integration. Our own findings, both as practitioners and researchers, support this claim; large industry development cases we have been in contact with in recent years tend to estimate spending some 10% of their total R&D effort building continuous integration and delivery capabilities, with one case of approximately 250 engineers estimating 25%. An internal survey conducted within a large software company we have previously studied further corroborates this. In it, four R&D units ranging from approximately 600 to 1200 headcount report continuous integration system headcount allocations varying from 5% to 7%.

Consequently we conclude that continuous integration and delivery expenditure in the industry is an important but under-researched field, but that there are strong indications that it is an area where significant effort is being spent. In addition to this, as software professionals ourselves, we witness an increasing incidence of customers requesting information on their suppliers' ability to continuously deliver and deploy new software, as that ability factors into their purchasing decisions.

The inherent paradox is evident: even though continuous integration and delivery capabilities are clearly believed by many industry companies — both producers and consumers of software solutions — to be crucial, and consequently invested heavily in, at the same time those same companies are unable or unwilling to approach the problem with the same engineering rigor and diligence as they do their commercial product development. In summary, we find that:

- Continuous integration and delivery capabilities are regarded as increasingly important in the industry.
- Vast amounts of money and resources are being expended in order to build continuous integration and delivery capabilities.
- Studied large scale continuous integration and delivery systems are bespoke in-house developed solutions.
- Despite its importance and the investments made, the problem of constructing effective and efficient integration systems isn't necessarily addressed in a systematic way.

3. Background

This section provides context by discussing the background of the work reported from in this paper.

3.1. Continuous integration modeling

In response to the situation outlined in Section 2, modeling and visualization techniques have been proposed [25,37] and ap-

Please cite this article as: D. Ståhl, J. Bosch, Cinders: The continuous integration and delivery architecture framework, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.11.006