Architecting for DevOps and Continuous Deployment

Mojtaba Shahin
CREST- The Centre for Research on Engineering Software Technologies
The University of Adelaide, Australia
mojtaba.shahin@adelaide.edu.au

ABSTRACT
Development and Operations (DevOps) in the context of Continuous Deployment (CD) have emerged as an attractive software development movement, which tries to establish a strong connection between development and operations teams. CD is defined as the ability to quickly put new releases into production. We believe that DevOps/CD brings new challenges for architects, which considerably impacts both on their (architectural) design decisions and their organizational responsibilities. We assert that there is an important and urgent need of sufficient research work to gain a deep understanding of how DevOps/CD adoption can influence architecting, architectural decision-making processes and their outcomes in an organization. This PhD research is aimed at understanding and addressing new challenges for designing architectures for supporting DevOps in the context of CD.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architecture

General Terms
Design

Keywords
Software architecture, DevOps, continuous deployment.

1. INTRODUCTION
With increasing complexity of software-intensive systems, the role of software architecture as a means of understanding and managing large-scale software intensive systems has been increasingly becoming important [1]. It has been recognized that different domains and contexts bring different challenges for architects. Hence, a given context can have considerable impact on architectural design decisions and organizational practices for architecting related activities and artifacts. Development and Operations (DevOps) in the context of Continuous Deployment (CD) is an emerging software industry movement to bridge the gap between development and operations teams [2]. CD is defined as the ability to frequently and reliably put new releases into production, with as much automation as possible [2, 11]. It is argued that an architect should make a software system's design as much simple and fine-grained as possible and try to remove those architectural elements that can be obstacle to deployment automation [11]. DevOps/CD practices necessitate an extensive use of infrastructure automation techniques, which can reduce the complexity of deployment and operations to a very large extent. Adopting and supporting DevOps/CD involve a large number of challenges because organizational processes and tools may not be ready to support highly complex and challenging nature of DevOps/CD. It is argued that one of the most pressing challenges which the organizations may encounter is how software applications should be (re-)architected to support DevOps practices such as Continuous Integration (CI), Continuous DElivery (CDE) and Continuous Deployment (CD) [2, 5]. A sound architecture helps ensure a desired level of quality attributes (e.g., deployability, testability, and loggability) and can enable short feedback cycle time including instant feedback from operations [3]. However, there has been a little research on what implications DevOps/CD can have on architecting [3, 5]. We assert that there is an important and urgent need of research to gain a deep understanding of how DevOps/CD adoption can influence the architecting processes and their outcomes in an organization. Therefore, this PhD research seeks architectural elements, practices, and patterns that support DevOps and continuous deployment.

ASWEC '15 Vol. II, September 28 - October 01, 2015, Adelaide, SA, Australia
ACM 978-1-4503-3796-0/15/09.
http://dx.doi.org/10.1145/2811681.2824996

2. RELATED WORK
Whilst there has been some research on the adoption of DevOps, continuous deployment (CD) and continuous delivery in the software development industry [5, 6, 9], there has been no systematic effort to explore the impact of DevOps and CD on software architecture. Claps at el. studied the technical and social challenges of adopting continuous deployment in a case software company and reported the mitigation strategies that had been adopted by the case company [6]. Team experience, continuous integration and infrastructure to name but a few have been reported as challenges of CD adoption. It has been reported in [5, 9] that continuous delivery and deployment provide the following benefits: (i) getting more and quick feedback from software development process and customers; (ii) having frequent and reliable release, which leads to improved customer satisfaction and product quality. To date, there are only two studies, which have explored the role of software architecture as a contributing factor when adopting CD and DevOps [3, 5]. Chen reported an experience of architecting 25 software applications for continuous delivery as well as proposed a set of architecturally significant requirements that should be effectively met in order to achieve the maximum benefits from continuous delivery [5]. Bellomo et al. have conducted an empirical study on three projects that had adopted continuous integration and continuous delivery [3]. The study concluded that most of decisions made to achieve the desire state of deployment (i.e., deployability) were architectural ones. The collected deployability goals and tactics from three projects have been used as building blocks for forming the deployability tactics tree.

3. RESEARCH DESIGN
This research is aimed at empirically building and evaluating a framework and associated tools for architecting to support DevOps/CD. Our main research questions (RQ) are as follows:

RQ1. What are the characteristics of DevOps/CD-amenable applications?

RQ2. How does (re-) architecting for DevOps/CD differ from that for non-DevOps/CD contexts?

RQ3. How can we codify (i.e., capture, document, and organize) the architectural practices and principles to make them useful for engineering DevOps/CD-amenable applications?

3.1 Research Methods

3.1.1 Systematic Literature Review

We use Systematic Literature Review (SLR) as one of the most widely used research methods in Evidence-Based Software Engineering (EBSE) paradigm [8]. The goal of SLR research method is to provide a well-defined process for identifying, evaluating, and interpreting all available evidence relevant to a DevOps and CD and it can help to establish solid background knowledge [8].

3.1.2 Exploratory Case Study

An empirical study should be carried out using a suitable research method chosen based on the nature of the studied problem and the research questions to be answered [7]. Since there is a little research on the impacts of DevOps/CD on architecting, we decided to carry out an exploratory case study in multiple small and large software development organizations. Case study is considered a suitable research method to investigate a contemporary phenomenon within its real-life context. Our study is an exploratory case study as it mainly deals with the "How" and "What" questions. Since this research has broad and high-level goals and there is little empirically gathered knowledge about architecting for DevOps/CD, we find an exploratory case study using interviews as data collection method appropriate. For example, survey can be followed up after the initial results of the interviews.

3.1.2.1 Data Collection Methods

We will use face-to-face interview as the main data collection method. However, in specific cases (e.g., travel restriction), other types of interview (e.g., Skype interview) may be employed. Since exploring the impact of DevOps/CD on software architecture is a new topic and we do not want to restrict the answers of interviewees in advance, we will do the semi-structured interviews, which involve open-ended questions. Participants will be recruited using purposive sampling in order to include practitioners who either have valuable experiences in (re-) architecting for DevOps/CD (e.g., architect) or are closely influenced by architecture (e.g., DevOps engineer). In addition to interviews, where possible we will gather data from documentations (e.g., architecture document) provided by participants to verify experiences and discussions shared by participants. Using both interviews and documentation (i.e., data triangulation) as data sources can increase the reliability of our study [13].

3.1.2.2 Data Analysis

The collected data (e.g., the interview transcripts) will be analyzed using thematic analysis method [4]. We will analyze the data to identify the implications of DevOps/CD on architecting, the challenges that the participants face when architecting for DevOps/CD and architectural practices (i.e., patterns and tactics) they employ. We plan to send the results of this study to the participants involved in the interviews. The benefits of this technique (i.e., member-checking) are twofold: (i) participants can strengthen the findings with further concrete and real examples; (ii) we can ask them if they agree with the findings and the preliminary assessment can be performed by them [10, 12]. For further validation, we want to organize a series of workshops with stakeholders involved in architecting activities in the context of DevOps/CD for getting their feedbacks on the initial findings from our study. The feedback from workshops can help us to verify and refine the findings of the study.

4. EXPECTED CONTRIBUTION

The expected outcomes of this PhD research are: (i) a systematic review of the existing research in the DevOps and continuous deployment paradigms; (ii) an evidence-based body of knowledge to support further development and adoption of DevOps/CD practices; (iii) identify and codify a list of architectural practices and patterns in the form of a framework and associated tools that should or should not be practiced when introducing DevOps and CD for complex applications.

5. ACKNOWLEDGMENTS

I would like to thank my supervisors Prof. Muhammad Ali Babar and Dr. Liming Zhu for their supports. This work is partially supported by NICTA. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

6. REFERENCES

[1] Bass, L., Clements, P., and Kazman, R. 2012. *Software architecture in practice*. Addison Wesley.

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

[3] Bellomo, S., Ernst, N., Nord, R., and Kazman, R. 2014. Toward Design Decisions to Enable Deployability: Empirical Study of Three Projects Reaching for the Continuous Delivery Holy Grail. In *Proceedings of 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 702-707.

[4] Braun, V. and Clarke, V. 2006. Using thematic analysis in psychology. *Qualitative research in psychology*. 3, 2, 77-101.

[5] Chen, L. 2015. Towards Architecting for Continuous Delivery. In *Proceedings of 12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 131-134.

[6] Claps, G.G., Svensson, R.B, and Aurum, A. 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology* 57, 21-31.

[7] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. 2008. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer and D.K. Sjøberg Eds. Springer London, 285-311.

[8] Kitchenham, B.A. and Charters, S. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report.

[9] Leppanen, M., Makinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mantyla, M.V., and Mannisto, T. 2015. The Highways and Country Roads to Continuous Deployment. *IEEE Software* 32, 2, 64-72.

[10] Martini, A. and Bosch, J. 2015. The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles. In *Proceedings of 12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 1-10.

[11] Newman, S. 2015. *Building Microservices*. O'Reilly Media, Inc.

[12] Stol, K.J., and Fitzgerald, B. 2014. Research Protocol for a Case Study of Crowdsourcing Software Development. Technical Report.

[13] Waterman, M.G. 2014. *Reconciling agility and architecture: a theory of agile architecture*. Doctoral Thesis, Victoria University of Wellington.