# Architectural challenges in continuous deployment

Olli Rissanen
Department of Computer Science
University of Helsinki
Helsinki, Finland
Email: olli.rissanen@helsinki.fi

***Abstract*—Currently more and more software companies are moving to lean practices, which often include shorter delivery cycles and thus shorter feedback loops. However, to achieve continuous customer feedback and to eliminate work that doesn't generate value, even shorter cycles are required. In continuous deployment the software functionality is deployed continuously at customer environment. This process includes both automated builds and automated testing, but also automated deployment. This adds more elements to the development pipeline, which often in a lean team consists of a version control system and a continuous integration server. Automating the whole process minimizes the time required for implementing new features in software, and allows for faster customer feedback. In this paper we review the architecture of existing software development pipelines that implement continuous deployment. We investigate the challenges a company typically faces during the transition to continuous deployment. We also explore the main issues and attributes required for the architecture of the the pipeline to allow gathering feedback in real-time. We aim for a coherent view of essential features required to build a development pipeline in such manner.**

## I. Introduction

Continuous deployment is an extension to continuous integration, where the software functionality is deployed frequently at customer environment. While continuous integration defines a process where the work is automatically built, tested and frequently integrated [1], often multiple times a day, continuous deployment adds automated acceptance testing and deployment. The purpose of continuous deployment is that as the deployment process is completely automated, it reduces human error, documents required for the build and increases confidence that the build works [2].

In an agile process software release is done in periodic intervals [3]. Compared to waterfall model it introduces multiple releases throughout the development. Continuous deployment, on the other hand, attemps to keep the software ready for release at all times during development process [2]. Instead of stopping the development process and creating a build as in an agile process, the software is continuously deployed to customer environment. This doesn't mean that the development cycles in continuous deployment are shorter, but that the development is done in a way that makes the software always ready for release.

It should also be made clear that continuous delivery differs from continous deployment. Both include automated deployment to a staging environment. Continuous deployment includes deployment to a production environment, while in continuous delivery the deployment to a production environment is done manually. The purpose of continuous delivery is to prove that every build is proven deployable [2].

In this paper we investigate the architectural challenges required for the software to enable continuous deployment. In an agile process software architecture is often ignored, citing You Ain't Gonna Need It (YAGNI) and big-up front design (BUFD) [4], but in a process with continuous deployment architecture is especially important.

As research on architectural challenges in continuous deployment is next to nonexistent, other areas of research have to be reviewed and applied to continuous delivery. Such areas are for example architectural challenges in agile development [4], analysis and management of architectural dependencies in iterative release planning [5], and the deployment production line [6]. Especially useful is also the research in importance of software architecture during release planning [7].

problems here

## II. Methods

As continuous deployment is a relatively new area of research, all aspects of it have not yet been investigated. To better understand the architectural challenges, a deeper insight on continuous deployment has to be gained.

## III. Results

## IV. Discussion

## V. Conclusion

## References

[1] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.
[2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.
[3] A. Cockburn, *Agile software development*. Addison-Wesley Boston, 2002, vol. 2006.
[4] P. Kruchten, "Software architecture and agile software development: a clash of two cultures?" in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 2. IEEE, 2010, pp. 497–498.
[5] N. Brown, R. L. Nord, I. Ozkaya, and M. Pais, "Analysis and management of architectural dependencies in iterative release planning," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*. IEEE, 2011, pp. 103–112.
[6] J. Humble, C. Read, and D. North, "The deployment production line," in *Agile Conference, 2006*. IEEE, 2006, pp. 6–pp.
[7] M. Lindgren, C. Norström, A. Wall, and R. Land, "Importance of software architecture during release planning." in *WICSA*, 2008, pp. 253–256.