UNIVERSITY OF MINNESOTA

SENG 5852

Continuous Integration, Delivery, & Deployment: Transforming the Software Industry

ANNOTATED BIBLIOGRAPHY

LUE XIONG

March 26, 2019

1 Bibliography

References

[1] Atkinson, B., & Edwards, D. (2018). Generic Pipelines Using Docker: The DevOps Guide to Building Reusable, Platform Agnostic CI/CD Frameworks. Berkeley, CA: Apress. doi: https://doi.org/10.1007/978-1-4842-3655-0

[2] Meyer, B. (2014). Agile! the Good, the Hype and the Ugly. Springer International Publishing. doi: https://doi-org.ezp3.lib.umn.edu/10.1007/978-3-319-05155-0

[3] Bosch, J. (2014). Continuous Software Engineering. Springer International Publishing. doi: https://doi-org.ezp1.lib.umn.edu/10.1007/978-3-319-11283-1.

Jan discusses the unprecedented evolution of software engineering to respond to rapid market changes and customer needs. The book and its studies emcompasses a group of collaborators from the academic software researchers and software industries called Software Center.

Stairway to Heaven conceptual model is introduced as a way to understand the way software industries transform to go against the competitive pressures of the market. The transition from more traditional development practices to more agile-centric development practices, resulting in continuous deployment of software to match the pace of the ever changing market. When moving to agile practices, Jan says there must be careful consideration of how it is introduced to an organization because it is a large organizational mindset shift. It requires an organization to have smaller development teams, cross-functional cooperation, and to think of software enhancements as pieces of small modular change as opposed to a larger system change.

Continuous integration is defined as having the ability to run automated test suite, source controlled codebase as a means of continual delivery, and system architecture that is modular in nature. However, continuous integration is found to be interpreted and implemented differently from company to company and with varying degrees of outcome. There are however similarities in all of the struggles for universal consensus of what it means to be continuously integrated. Inherent in agile practices is the notion of self-healing — using a collection of metrics such as static analysis, visual analytics and business intelligence to combat software

issues. These metrics are ran in an automated environment, capturing data needed to take autonomous action for the purpose of recovering from software abnormalities.

As a result of the culture shift to an agile environment in the software industry, traditional leadership roles have dramatically changed from a command and control style to more engaging and supportive approach. Project managers, even if skilled, are often left to find their own role in the fast-paced environment of agile culture, or face leaving an organization altogether. The key to any transformation to a new culture Jan says, is leadership culture change.

[4] Continuous Delivery, Deployment & Integration: 20 Key Differences. (2018, June 04). Retrieved from `https://stackify.com/continuous-delivery-vs-continuous-deployment-vs-continuous-integration`

[5] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access, 5, 3909-3943. doi: `10.1109/access.2017.2685629`

[6] Ståhl, D. (2017). Large Scale Continuous Integration and Delivery: Making Great Software Better and Faster. [Groningen]: University of Groningen.

Daniel Stahl's doctoral thesis explores and investigates the changing culture of software engineering in response to the increasingly software-dependent societies we live in. He breaks up his thesis about continuous integration and delivery into three main parts. The first part focuses on the nature of continuous integration, how it is interpreted, and how it is implemented by different software entities. Daniel speaks to the fact that continuous integration is rooted from a software development practice called eXtreme programming — an agile practice created by Kent Beck. The idea of eXtreme programming is to address software integrated changes early and often, of which continuous integration as a part of the practice aims to aid in developer productivity, agile testing, communication, and project predictability. Daniel states that the case studies of continuous integration benefits by relative literature are merely suggestions, rather than case studies backed with evidence. In an effort to provide evidence for these claims, he presents evidence with a multi-case study of large-scale development projects that have software professionals engaged in continuous integration as daily practice. Explored

also in the first part the doctoral thesis includes the discussion of continuity and scalability of continuous integration. As project codebase grows larger, test scope does as well; resulting in longer build compilation and test running time. Daniel has found that there is a correlation between the size of a software organization and its' continuous integration tendencies.

The second part focuses on system modeling of continuous integration and delivery solutions. Daniel attempts to close the gap of ambiguity for software professionals and academic researchers in describing and designing continuous integration. Daniel employs continuous integration modeling techniques in a multi-case study and analyzes the effects of how software professionals understand their own continuous integration systems.

The third part focuses on the issue of traceability with continuous integration system requirements. Traceability has long been known to the software industry as a practice overwhelmed with its share of struggles. It is also a practice that is of considerable importance to software engineering. In order to weave through the software industry struggle, Daniel attempts to describe and evaluate possible solutions in context of continuous integration systems.

An observation of Daniel Stahl's work on continuous integration and delivery seems to suggests that he implicitly differentiates between continuous 'integration' and 'delivery'. He often uses the term 'continuous integration' and has a seemingly intentional habit to exclude the term "delivery". It is a little concerning that he does not explicitly speak to the differentiation though.

[7] Ståhl, D., Hallén, K., & Bosch, J. (2016). Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. Empirical Software Engineering, 22(3), 967-995. doi:
10.1007/s10664-016-9457-1