UNIVERSITY OF MINNESOTA
SENG 5852

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

RESEARCH PAPER

LUE XIONG

March 23, 2019

Contents

1 Introduction

The software industry is transforming at a rapid pace to accommodate the dynamic nature of the market and as a result, it continues to struggle to find process-identity with continuous software engineering. Software engineering has for two decades, experimented with the concept of distributing software in faster release cycles; endeavoring to do so without sacrificing reliability and security. To achieve such a goal, there has been a widespread movement in the technical community to advocate for using Agile practices, and in particular: continuous integration, delivery, and deployment. The traditional methods of software development no longer meet the need of businesses that – now more than ever – want to proactively engage and retain their customers. The organizational transition to Agile practices demands a large mentality change and require individuals to recognize software as incremental features developed with cross-collaboration of small comprehensive team units, as opposed to large modules developed by siloed units.

2 Body

2.1 What is Continuous Integration, Delivery, & Deployment

2.1.1 Inherently Agile

- What is Agile?
- What are the core ideas of Agile?
- How does Agile tie in with CI/CDE/CD?

2.1.2 Continuous Integration

- What does CI mean?

2.1.3 Continuous Delivery

- What does CDE mean?

2.1.4 Continuous Deployment

- What does CD mean?

2.2 Differences of Interpretation & Implementation

2.2.1 Viewpoint of Software Professionals

- How do software professionals interpret and implement CI/CD/CDE?

2.2.2 Viewpoint of Academic Researchers

- How do academic researchers interpret and believe how CI/CD/CDE should be implemented?

2.2.3 Collaboration Effort

- What effort is there to bridge the phenomena of non-collaboration between developers and researchers?

2.3 Benefits of Continuous Integration, Delivery, & Deployment

2.3.1 Self-healing Systems

- What are the metrics and tools that software professionals use to mitigate having to manually fix software issues?
- How do these self-healing systems work?

2.3.2 Reduce Risk

- How does continuous software engineering reduces risk in systems?

2.3.3 Faster Release Cycles

- How are faster release cycles are achieved?

2.3.4 Overall Cost Reduction

- Why will all of the above will reduce cost?

2.4 Struggles of Traceability

2.4.1 Importance

- What is the importance of traceability for the software engineering community?

2.4.2 Problem of Mapping

- What is the problem of mapping requirements to implemented code and the converse?

2.4.3 Eiffel Framework

- What is the proposed solution to address traceability issues in CI/CDE/CD environments?

2.5 Transition an Agile Environment

2.5.1 The Effect of Organizational Change to Agile

- What are the problems that businesses face in attempt to switch to Agile practices?

2.5.2 Roles in Agile

- What are typical roles that each individual plays in an Agile environment?
- Why do these roles exist?

2.5.3 Paradigm Shift in Leadership

- How has leadership changed as a result of Agile?

3 Conclusion

3.1 Rephrase Thesis Statement

3.2 Closing Statement

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

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

[4] 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

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

[6] 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