An investigation into the effectiveness and impact of a Continuous Delivery pipeline upon university-level games development teams?

Frost Donovan

***Abstract*—What's the problem? What am I looking at? How does that help solve the problem?**

Opening, Challenge, Action, Resolution

Continuous Delivery (CD) is a technique designed to increase reliability and consistency with delivery of builds. This can then help to increase frequency of testing, the accountability of the development team, as well as the teams transpareny and *"all being on the same page ness"*. This can increase team moral as well as stakeholder confidence, as both are able to regularly see the current state and rate of progress for the project. This encourages regular analysis of development pace and project scope, both internal and external.
This stakeholder confidence and awareness of scope is a common problem within student development teams, so this investigation will answer to what extent the deployment of a CD pipeline will help to reduce these problems.

I. Introduction

A. What is Continuous Delivery?

CD is an expansion of Continuous Integration (CI), a pipeline for the continuous integration of code, being developed on separate branches, into the main branch. This code is merged into the main branch and then tested to ensure there are no merge conflicts or obvious errors thrown from the combined code. Assuming all of these tests pass, this merged code is pushed to the project repository. A key part of this pipeline is that the entire process is automated, ensuring minimal time is wasted waiting for tests to run or code to be uploaded. [1], [2]
CD takes this one step further, in that after a project passes through the CI pipeline, the code is then compiled, packaged, and further tests that require the project to be compiled can be run. These tests should not be run before other tests that don't require compilation due to Fail Fast principles [3], [4]. Assuming there are no failures during compilation or testing, the built program can then be uploaded to somewhere where it can be easily accessed. Here, it is important to make a distinction between Continuous Delivery, where the build is available internally but not to users, and Continuous Deployment, where the build is pushed straight out to the current product users. core to agile methodology [5]

B. Benefits & Drawbacks of CD

The primary benefit of CD is reduced cycle time - a reduction in the time it takes for a change in the project to happen and then for the user to have that change applied to their version of the software. This principle is core to the agile methodology, being the very first principle in the agile manifesto [5]. This faster cycle time means that feedback from active users can be obtained much quicker, both on the effectiveness of bug fixes and also on new features. Agile is designed to avoid the pitfalls of Waterfall [6], one of which is the commitment of significant time and/ or resources into features that either are unattainable, or not actually wanted by the user. While other agile methods, such as Scrum or Extreme Programming [7], [8], can be an important part of the agile process, the effectiveness of a development team in delivering *value* is always going to be dependant on the speed and reliability with which user feedback can be obtained.
A benefit of this fast cycle time is not only are players able to see these changes faster, but stakeholders and publishers are able to see development progress, both regularly and on demand. This can be a significant step to building trust between a development studio and publisher, especially if the studio is new or doesn't have an existing relationship with the publisher [9].
Part of the CD pipeline is testing, with a suite of unit tests being run on the code as part of the build process. As this build process is run consistently, rather than all at once leading up to a main release, this means bugs are found incrementally, stopping the accumulation of technical debt, reducing the cost to fix bugs, and reducing stress on programmers by preventing an overwhelming influx of bugs. While these unit tests will likely catch a lot of bugs, some bugs will only be caught during human playtesting. This decreased cycle time means that human playtesting can happen sooner & more regularly, and fixes are delivered to testers & players almost immediately, rather than having to wait for the next release window.

Another strength of a Continuous Delivery pipeline is that it is fully automated. This allows less developer time to be spent setting up build or test environments and manually going through the build process, and more time on actually creating the product. This can be a *significant* time save, with some large scale projects reportedly taking weeks to set up environments ready to produce a release build [1], [10]. This system also significantly reduces the chance that there are any errors caused by mistakes during the build process as this build-release pipeline will have had many iterations of the product pass through it, before a major release deadline.

This increases the reliability and stability of new releases.

II. Background & Supporting Literature

With these benefits in mind it raises two questions; If a CD pipeline is this important and valuable, is it being taught to new developers in further education? If it is, is it actually as effective in practice with student teams as it is in theory?

Has this been done before in academic setting [11]–[14]? Links to other things - CI [2], Unit tests, regular product reviews, stakeholder (supervisor) confidence, git flow [15] Best practices [16]?

III. Research Question

From the above sources, I have formed **The actual question**

A. hypothesis & null hypothesis

IV. Research Methodology

A. Experimental Design

B. Limitations

Time, resources

C. Sampling Plan

Sample size, sampling method

D. Data management plan

Managing, collecting, & storing data

E. Data Analysis

T-test?

F. Ethical Considerations

V. Appendix

Data analysis code, supporting screenshots, list of unit tests & testing plan

VI. Artifact

A. What will be made

CD pipeline utilising Github Actions. Tool to set up secrets? Would be sick https://docs.github.com/en/rest/reference/actions#secrets
Continuous delivery or continuous deployment? Scope as deployment would need to include itch integration w/ butler, although it looks relatively simple to set up. Autoupload to steam & itch! https://itch.io/docs/butler/

B. How will I ensure Quality

Quality control. Roadmap? Unit Testing? Integration testing?

C. How will I create it

D. Why will this answer the questions

References

[1] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[2] S. Pittet, "Continuous integration vs. continuous delivery vs. continuous deployment."

[3] J. Shore, "Fail fast [software debugging]," *IEEE Software*, vol. 21, no. 5, pp. 21–25, 2004.

[4] Atlasian, "Bamboo best practice - using stages," 2021.

[5] K. B. M. B. A. van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas, "Manifesto for agile software development."

[6] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, pp. 328–338, 1987.

[7] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods.," *Adv. Comput.*, vol. 62, no. 03, pp. 1–66, 2004.

[8] C. Keith, *Agile Game Development with Scrum*. Upper Saddle River, NJ : Addison-Wesley, 2013.

[9] M. Futter, *The gamedev business handbook: how to build the business you'll build games with*. London: Bithell Games, 2017.

[10] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.

[11] S. Krusche and L. Alperowitz, "Introduction of continuous delivery in multi-customer project courses," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, (New York, NY, USA), p. 335–343, Association for Computing Machinery, 2014.

[12] S. Krusche and B. Bruegge, "User feedback in mobile development," in *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle*, MobileDeLi '14, (New York, NY, USA), p. 25–26, Association for Computing Machinery, 2014.

[13] B. Bruegge, S. Krusche, and M. Wagner, "Teaching tornado: From communication models to releases," in *Proceedings of the 8th Edition of the Educators' Symposium*, EduSymp '12, (New York, NY, USA), p. 5–12, Association for Computing Machinery, 2012.

[14] K. Kuusinen and S. Albertsen, "Industry-academy collaboration in teaching devops and continuous delivery to software engineering students: Towards improved industrial relevance in higher education," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 23–27, 2019.

[15] V. Driessen, "A successful git branching model," 2012.

[16] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.