**CASE STUDY**

## Continuous Integration at Bazaarvoice (2012)

Ernest Mueller, who helped engineer the DevOps transformation at National Instruments, later helped transform the development and release processes at Bazaarvoice in 2012.[14] Bazaarvoice supplies customer-generated content (e.g., reviews, ratings) for thousands of retailers, such as Best Buy, Nike, and Walmart.

At that time, Bazaarvoice had $120 million in revenue and was preparing for an IPO.[‡] The business was primarily driven by the Bazaarvoice Conversations application, a monolithic Java application comprising nearly five million lines of code dating back to 2006, spanning fifteen thousand files. The service ran on 1,200 servers across four data centers and multiple cloud service providers.[15]

Partially as a result of switching to an Agile development process and two-week development intervals, there was a tremendous desire to increase release frequency from their current ten-week production release schedule. They had also started to decouple parts of their monolithic application, breaking it down into microservices.

Their first attempt at a two-week release schedule was in January of 2012. Mueller observed, "It didn't go well. It caused massive chaos, with forty-four production incidents filed by our customers. The major reaction from management was basically 'Let's not ever do that again.'"[16]

Mueller took over the release processes shortly afterward, with the goal of doing biweekly releases without causing customer downtime. The business objectives for releasing more frequently included enabling faster A/B testing (described in upcoming chapters) and increasing the flow of features into production. Mueller identified three core problems:[17]

- Lack of test automation made any level of testing during the two-week intervals inadequate to prevent large-scale failures.
- The version control branching strategy allowed developers to check in new code right up to the production release.
- The teams running microservices were also performing independent releases, which were often causing issues during the monolith release.

Mueller concluded that the monolithic Conversations application deployment process needed to be stabilized, which required continuous integration. In the six weeks that followed, developers stopped doing feature work to focus instead on writing automated testing suites, including unit tests in JUnit, regression tests in Selenium, and getting a deployment pipeline running in TeamCity. "By running these tests all the time, we felt like we could make changes with some level of safety. And most importantly, we could immediately find when someone broke something, as opposed to discovering it only after it's in production."[18]

They also changed to a trunk/branch release model, where every two weeks they created a new dedicated release branch, with no new commits allowed to that branch unless there was an emergency—all changes would be worked through a sign-off process, either per-ticket or per-team through their internal wiki. That branch would go through a QA process, which would then be promoted into production. The improvements to predictability and quality of the releases were startling:[19]

- **January 2012 release**: forty-four customer incidents (continuous integration effort begins)
- **March 6, 2012 release:** five days late, five customer incidents
- **March 22, 2012 release:** on time, one customer incident
- **April 5, 2012 release:** on time, zero customer incidents

Mueller further described how successful this effort was:

> We had such success with releases every two weeks, we went to weekly releases, which required almost no changes from the engineering teams. Because releases became so routine, it was as simple as doubling the number of releases on the calendar and releasing when the calendar told us to.
>
> Seriously, it was almost a non-event. The majority of changes required were in our customer service and marketing teams, who had to change their processes, such as changing the schedule of their weekly customer emails to make sure customers knew that feature changes were coming.
>
> After that, we started working toward our next goals, which eventually led to speeding up our testing times from three plus hours to less than an hour, reducing the number of environments from four to three (Dev, Test, Production, eliminating Staging), and moving to a full continuous delivery model where we enable fast, one-click deployments.[20]

***By systematically identifying and addressing three core problems, this case study illustrates the power of practices like feature freezes (in this case to work on automated testing) and using trunk-based development to enable small batch sizes and accelerate release cycles.***