Stanford eCorner

An Argument for Continuous Deployment

Eric Ries, *Author*

September 30, 2009

Video URL: http://ecorner.stanford.edu/videos/2294/An-Argument-for-Continuous-Deployment

Author and entrepreneur Eric Ries stresses continuous deployment - that is, the updating of code and website changes as frequently as every twenty minutes - as a necessary asset to the functioning of a lean startup. He states that all online product development and engineering changes should be implemented slowly and with immediate testing for each small change. Working in small batches allows for the immediate catching of errors, quick team response, reduced implementation of resources, and more control and flexibility over the finished product. And along the way, human intervention also builds up the intelligence and capabilities of the in-house testing and deployment systems. Ries also offers specific applications and engineering functions to make continuous deployment possible.

Transcript

Here's continuous deployment. That's what I talked about before - that 20 minutes between checking in code and having it live in production. So here's the magic that allows you to do that without, you know, taking the site down. In order to do continuous deployment, you have to be able to like actually physically deploy the software quickly, like I said -- about 20 minutes. Most of that 20 minutes is actually not the mechanical process of doing the deployment. It's trying to certify that this change is non-harmful to the business. So just to give you an example: Let's say, like, it's April Fool's Day and you were back at my old office at IMVU. So, hey, this would be hilarious. Let's check in, I don't know, an infinite loop to the front page of IMVU.com so it doesn't work anymore. And let's deploy that because that would be hysterical.

Right? OK. You know we go back to my desk; we'd make the change; we try that on my local sandbox; it would blow up; we check it in. And sometime in the next 20 minutes, I would get an email from the cluster effectively: "Dear Eric, thank you so much for attending to check in that change. It's a terrible idea, and it has automatically been reverted. And not only that, we've notified everybody in the team that this happened and that something has gone wrong and nobody else can make a change until someone gets to the bottom of what has happened." We, of course, had to be able to do that revert quickly; and that's got to be true, not just at the level of like unit test and automated test because the example I just gave is kind of easy. OK, you know, if I make the whole site not work -- like, we can have tests that catch that. Let me give you a better example. This is actually a real example. Instead of doing that, I'm going to change the checkout button in the e-commerce flow for IMVU. I'm not going to review it because the test would catch that.

I'm just going to make it white on white so it's invisible to humans. OK? So now we're going from having a business to having a hobby -- not a good change. And again, the same thing would happen. In 20 minutes, that thing would be automatically reverted because the system has the intelligence necessary to know that we've made a change that is going to impact our business metrics in a catastrophic way. This idea of shutting down the line when it happens is really important. This is kind of by metaphor to the Toyota production system. Everyone has heard that story of the andon cord, the cord that any worker can pull to indicate that there's a quality problem that stops the whole production line. Even though it's very expensive to have the whole production line stopped, it's even more expensive to let a quality problem kind of fester and proceed down the line. So we use the same idea at a lean startup. So anytime we have a test failing or a regression that's hit production, we want to stop everything and create the space for human beings to get to the root cause of what happened.

In order for this all to work, we have to work in very small batches. So just for the engineers in the room: We have one engineer working on a project, say, for three days without checking in and deploying. We would consider that to be an extremely large batch. It will kind of be like, what's going wrong here? We certainly never have a whole team of five working on something for two weeks all by themselves generating integration risk -- no way. And that means we have to break large projects down into small batches, which is actually a lot easier than it sounds because 80% of the work of any new feature is actually all the, like deep interchanges we have to make to reconcile that feature with all the other features in the product -- make sure we didn't break anything. So there's just hundreds of little changes that are all supposed to be without side effects, right? And our theory is: If you make a change, and it supposedly has no side effects, let's deploy that change right now and be sure. Here's what it actually looks like from a technical point of view, for those who are technical in the room, you know what I'm talking about. For everybody else, these are some key words you can take back to your startup someday and say, hey, are we doing any of these things? We run all our tests locally in a sandbox. Everyone in the company has the ability to run a simulated version of the product on their own desktop. We do continuous integration using a service called Buildbot that means we have unit tests, functional tests, acceptance test -- every kind of automated test you can imagine.

We do that process of incremental deploy - that's monitoring the health of our product as we're doing the deployment to see if there's a problem so we can do -- revert it fast. And, of course, we do real time -- what we call predictive monitoring which allows us to get a human being up in the middle of the night if we have a catastrophic problem. But here is the most important things we do: Whenever somebody sees a failure, we want to get to the root cause of why. So we're going to ask: How come our defense mechanisms, the cluster immune system, didn't catch this? And every time we fail to catch it, we're going to make our defenses a little bit stronger. That's the essence of continuous deployment.