engineering.imvu.com

IMVU's Approach to Integrating Quality Assurance with Continuous Deployment

7-9 minutes

We've heard a lot of interest from folks we've talked to in the tech community and at conferences about our Continuous Deployment process at IMVU, and how we push code up to 50 times a day. We've also received some questions about how we do this without introducing regressions and new bugs into our product, and how we approach Quality Assurance in this fast-paced environment.

The reality is that we occasionally do negatively impact customers due to our high velocity and drive to deliver features to our customers and to learn from them. Sometimes we impact them by delivering a feature that isn't valuable, and sometimes we introduce regressions or new bugs into our production environment.

But we've delivered features our customers didn't like even when we were moving more slowly and carefully—and it was actually more costly because it took us longer to learn and change our direction. For example, we once spent more than a month of development time working on a

feature called Updates–similar to the Facebook “friend feed”, and we guessed wrong–way wrong–about how our customers would use that feature. It took us a way too long to ship a feature nobody actually wanted, and the result was that we delayed delivery of a feature that our customers were dying to have: Groups.

Asking customers what they want takes guesswork and internal employee opinions out of product development decisions, making it easy to resolve questions such as, “Should we build tools to let users categorize their avatar outfits, or should we build a search tool, or both?” We make the best decisions we can based on available customer data, competitive analysis, and our combined experience and insights—then build our features as quickly as possible to test them with customers to see if we were right.

We’ve found that the costs we incur–typically bugs or unpolished but functional features–are worthwhile in the name of getting feedback from our customers as quickly as possible about our product innovations. The sooner we have feedback from our customers, the sooner we know whether we guessed right about our product decisions and the sooner we can choose to either change course or double down on a winning idea.

Does that mean we don’t worry about delivering high quality features to customers or interrupting their experience with our product? Nothing could be further from the truth.

For starters, we put a strong emphasis on automated testing. This focus on testing and the framework that

supports it is key to how we've structured our QA team and the approach they take. Our former CTO and IMVU co-founder Eric Ries has described in detail the infrastructure we use to support Continuous Deployment, but to summarize, we have implemented:

- A continuous integration server (Buildbot) to run and monitor our automated tests with every code commit
- A source control commit check to stop new code being added to the codebase when something is broke
- A script to safely deploy software to our cluster while ensuring nothing goes wrong. We wrote a cluster immune system to monitor and alert on statistically significant regressions, and automatically revert the commit if an error occurs)
- Real-time monitoring and alerting to inform the team immediately when a bug makes in through our deployment process
- Root cause analysis (Five Whys) to drive incremental improvements in our deployment and product development processes

This framework and our commitment to automated testing means that software engineers write tests for everything we code. We don't have a specialized role that focuses solely on writing tests and infrastructure–that work is done by the entire engineering team. This is one factor which has allowed us to keep our QA team small. But you can't catch all regressions or prevent all bugs with automated tests.

Some customer use cases and some edge cases are too complex to test with automation. Living, breathing, intelligent people are still superior at doing multi-dimensional, multi-layered testing. Our QA Engineers leverage their insight and experience to find potential problems on more realistic terms, using and testing features in the same ways real customers might use them.

Our QA Engineers spend at least half their time manually testing features. When we find test cases that can be automated, we add test coverage (and we have a step in our Scrum process to help ensure we catch these). We also have a Scrum process step that requires engineers to demonstrate their features to another team member– essentially, doing some basic manual testing with witnesses present to observe. Since we have far more features being built than our QA Engineers have time to test, it also forces the team to make trade-offs and answer the question, “What features will benefit most from having QA time?” Sometimes this isn’t easy to answer, and it forces our teams to consider having other members of the team, or even the entire company, participate in larger, organized testing sessions. When we think it makes sense, our QA Engineers organize and run these test sessions, helping the team to find and triage lots of issues quickly.

Our QA Engineers also have two more important responsibilities. The first is a crucial part our Scrum planning process, by writing test plans and reviewing them with product owners and technical leads. They help ensure

that important customer use cases are not missed, and that the engineering team understands how features will be tested. Put another way, our QA engineers help the rest of the team consider how our customers might use the features they are building.

The second responsibility is what you might expect of QA Engineers working in an Agile environment: they work directly with software engineers during feature development, testing in-progress features as much as possible and discussing those features face-to-face with the team. By the time features are actually “ready for QA”, they have usually been used and tested at some level already, and many potential bugs have already been found and fixed.

Regardless of how much manual testing is completed by the team before releasing a feature, we rely again on automation: our infrastructure allows us to do a controlled rollout live to production, and our Cluster Immune System monitors for regressions, reducing the risk of negatively impacting customers.

Finally, once our features are live in front of customers, we watch for the results of experiments we’ve set up using our A/B split-test system, and listen for feedback coming in through our community management and customer service teams. As the feedback starts rolling in—usually immediately, we’re ready. We’ve already set aside engineering time specifically to react quickly if bugs and issues are reported, or if we need to tweak features to make

them more fun or useful for customers.

We’re definitely not perfect: with constrained QA resources and a persistent drive by the team to deliver value to customers quickly, we do often ship bugs into production or deliver features that are imperfect. The great thing is that we know right away what our customers want us to do about it, and we keep on iterating.