Copy of Sample Software Engineering Project

by Devmountain Tutorials

WHAT'S COVERED

This section will explore a sample software engineering project by discussing:

1. FIRST RELEASE
2. PREVENTING BUGS

1. FIRST RELEASE

Today is a big day at Poodle Jumper! Our team is preparing to release an iteration we've been working on for the past two months. The release provides a new feature for customers to schedule recurring services for their pets so they don't have to schedule it manually every time.

This new feature required a few architectural changes, so we'll need to bring the services down to make the changes. When this kind of deployment strategy is required, we do the work in the middle of the night to avoid interruption to our customers and providers.

We schedule it in advance and let the customers and support team know when they can expect the service to be back up and running. Before the release, I capture snapshots of the database and applications to make sure we have backups just in case something goes wrong.

Tonight, our release is scheduled for midnight. Everyone from the release team joins the virtual meeting from home. It starts by bringing down the service and updating the code and database to the new version. This can be as fast as five minutes or as slow as several hours, depending on the changes.

When the new version is in place, the automated tests run and the team splits up the manual test cases to make sure it is working as expected. I keep a really close eye on the error logs and analytics to capture anything unexpected.

If we find major issues that can't be fixed quickly, the release is rolled back. Luckily, tonight the release is a success. We'll get a few hours of sleep before logging in tomorrow to make sure everything is still working well. I'm on-call, so I know that if anything comes up, I'll get a call.

The next morning, I come in and immediately look at the analytics and support queue. I know nothing serious happened because I didn't get any alerts after the release, but I see one new issue the support team sent to us.

Amanda (QA) has taken the initiative to look at it and provided me with the details. When the user tries to schedule a recurring service for her dog, she sees an error message that says, "An unexpected error has occurred." Amanda looked at the user's account and saw the same error when she tried to schedule as the user.

Being able to reproduce an error is extremely helpful for resolving it. We leveraged tools to debug the error and identified an issue with the data in the user's account.

We call the environment users to interact with Production. It is bad practice to test with production data because it can negatively impact users. To avoid this, I make a clone the data and put it in a test environment. This will let us try the fix and verify it works. I run a database command to change the data for the user to the expected format. Amanda and I try to reproduce the problem, but the error message no longer displays, and the feature is working as expected.

I'm relieved to have figured out the problem for this one user, but I need to dig a little deeper. Is it possible that other users will experience this as well?

I run a query, or a search, in the database to identify if anyone else has the same issue with their data. The query returns 143 additional users in the bad state. I'm able to fix their account with the same command in the test environment. Amanda and I have a high level of confidence in the fix, so I execute the command in production and let the support team know we've resolved the issue.

2. PREVENTING BUGS

It may sound like the job is done and it is time to move on, but I want to circle back and make sure we've put in mechanisms to prevent users from getting into this bad state in the future.

When a small number of users see an error, it is easy to blame them because the feature works for everyone else, but there is almost always an underlying issue that allows that error to happen. Preventing issues from happening and fixing them so that they stay fixed is important to the quality of our software.

Not all bugs or issues are solved so quickly. If we can't reproduce the issue, it is hard to know what code to change and almost impossible to verify if it is resolved. There are times I find issues in our logs, and I have no idea how it happened.

It can be really frustrating when I don't have enough information to fix something. That is why I'm grateful for our users who report errors that they see. We built a feature in the app so it is easy for users to send us issues and feedback. But we do our best to leverage analytics and monitoring, so users don't have to tell us when our system breaks.

Every week I spend a portion of my time thinking about where things will break and creating fault-tolerant systems. Fault-tolerant means building something, so it won't completely break when an error or something unexpected happens.

You'd be surprised at the weird things that can break an application. I've seen large organizations come to a screeching halt from a single typo. You may feel like you see technology outages all the time, and you're right. They happen even to mature organizations. Computers are amazingly precise, and they can execute with amazing accuracy–when the humans programming them are equally precise.

At the end of the day, everyone on the development team is responsible for ensuring quality in our software. It takes a team effort to make this happen.
