Operational Excellence in April Fools' Pranks

**BEING FUNNY IS
SERIOUS WORK.**

THOMAS A. LIMONCELLI

At 10:23 UTC on April 1, 2015, stackoverflow.com enabled an April Fools' prank called StackEgg.¹ It was a simple Tamagotchi-like game that appeared in the upper right corner of the website. Though it had been tested, we didn't account for the additional network activity it would generate. By 13:14 UTC the activity had grown to the point of overloading the company's load balancers, making the site unusable. All of the company's web properties were affected. The prank had, essentially, created a self-inflicted denial-of-service attack.

The engineers involved in the prank didn't panic. They went to a control panel and disabled the feature. Network activity returned to normal, and the site was operating again by 13:47 UTC. The problem was diagnosed, fixed, and new code was pushed into production by 14:56 UTC. The prank was saved!

Was Stack Overflow lucky that the engineers had designed the prank so that it could be easily disabled? No, it wasn't luck. It was all in the playbook for operational excellence in AFPs (April Fools' pranks).

A successful AFP depends on many operational best practices. In this column, I'll share some of the key ones.

WHAT MAKES AN APRIL FOOLS' PRANK FUNNY?
Before discussing the technical details, let's look at

what makes an AFP funny. The best AFPs are topical and absurdist.

Topical means it refers to current events or trends. This makes it relevant and “a thinker.” Topical would be displaying your website upside-down after a large and highly publicized acquisition by a major Australian competitor. (Australians tell me that kind of joke never gets old.) Doing that to your website otherwise just announces that your web developers finally read that part of the CSS3 spec.

Secondly, it must be so absurd that it reveals a hidden truth. Absurdist humor is not simply silly for silliness’ sake. Absurdism acts as a crucible that burns away all lies to get to the truth.

Stack Overflow’s 2017 prank, “Dance Dance Authentication,” was both topical and absurdist.³ The prank was a blog post and accompanying demonstration video for Stack Overflow’s new (fictional) authentication system. Rather than the usual 2FA (two-factor authentication) system that requires an authenticator app or key fob, this system required users to turn on their webcams and dance their password. This was topical because recent growth in 2FA adoption meant many Internet users were experiencing 2FA for the first time. It was absurdist because it took the added burden and nuisance of 2FA to an extreme. It revealed the truth that badly implemented security sacrifices convenience.

Inspiration for absurdity should come from reality. For example, the Go programming language is an intentionally minimalistic language—a reaction against bloated languages such as C++ and Java. It seems like every C++

A prank should not get in the way of business or harm customers.

or Java programmer who learns Go posts to forums demanding dozens of features that are “missing.” This leads to a discussion about why those features are intentionally missing from Go. This discussion seems to happen on a weekly basis. A good AFP for Go would be a blog post announcing that Go 2.0 will include all those “missing” features and, in fact, they have been implemented and are ready for use. The article would then link to the download page for Java.

WHAT MAKES AN APRIL FOOLS’ PRANK UN-FUNNY?

A prank should not get in the way of business or harm customers. For example, a 2016 Gmail prank called “Drop the Mic” gave users a button that would send a farewell message to someone, then block all email from that person... forever. There was no “Are you sure?” prompt. As you can guess, this disrupted actual customers trying to do actual business.² Google disabled the prank a few hours later.

An AFP should not mock a particular person (that’s just mean) or group of people (that’s just hateful). The exception to this is that it is always OK to mock people more powerful than you. Punch up, not down.

- ➔ **Punch up:** mock elite people who don’t realize how privileged they are; mock the CEO who bragged he’s saving the company money by using his private jet.
- ➔ **Don’t punch down:** do not mock the less fortunate—for example, don’t mock homeless people or any group of powerless people in society; racist, sexist, or homophobic humor is not funny because it is inherently punching down.

An AFP should be funny to the audience, not just the

people who created it. Every year plenty of companies produce AFPs that fall flat because they are inside jokes that everyone in the company finds hiii-larious. That's all well and good, but if the intention was to make customers laugh, it really shouldn't depend on them knowing that Larry in accounting loves World of Warcraft.

As with any feature, user acceptance testing should be done with a wide variety of users. Be sure to include some nonusers. You might consider doing user experience testing, but since most companies don't, why start now?

ENGINEER IT LIKE ANY OTHER FEATURE

The end-to-end process of creating and launching the prank should be the same as any other feature. It should start with a concept, then have a design and execution plan, launch plan, and operational runbook. Involve product management. Have requirements, specifications, a project schedule, testing, and so on. If it is a big prank, beta testing with users sworn to secrecy may be required.

Like any major feature, the earlier you involve operations, the better. Operations' worst nightmare is to be told that a major feature is being launched tomorrow... "Would you please set up 10 new servers and find a petabyte of disk space?" April Fools' pranks are no different. They often require extra bandwidth, isolated servers, firewall rules, and other tasks that take days or weeks to complete.

FEATURE FLAGS

The prank should be easy to enable and disable. Hide the feature behind a "feature flag." With the flag off, the

feature is in the code but dormant. Enabling the prank in production is a matter of turning the flag on. Disabling it is a simple matter of turning the flag off. Developers can test the feature by enabling the flag in the development and test environments. Some flag systems can automatically be on for certain user segments.

Some companies can launch or disable a feature only by rolling out new code into production. This is bad for many reasons. It is riskier than feature flags: if the release that removes a prank is broken, do you revert to the previous release (with the prank) or the prior release (which may be too old to deploy into production)? Code pushes are difficult to coordinate with PR, blog posts, and so on: they might take minutes or hours, not seconds, like flipping a feature flag. Code pushes require more skill: in many environments, code pushes are done by specific people, who might be asleep. In an emergency you want to empower anyone to shut off the prank. The process should be quick and easy. Lastly, if the prank has overloaded the network, it may also affect the systems that push new code. Meanwhile, a feature “flag flip” is simpler and more likely to just plain work.

The way you structure an AFP project is unusual in that the deadline cannot change. There are three levers available to managers: deadline, budget, and features. If a project is going to be late, management must adjust one of those three. An AFP, however, cannot adjust the deadline and usually has a limited budget. Therefore, it is important to segment the features of the prank. First implement the basic prank, then add “would be nice” features. As you get closer to the deadline, throw away the

less important features. When a badly structured prank is late, all features will be 80 percent done, which means 0 percent of them can be launched. You blew it. When a well-structured prank is late, 80 percent of the features are ready to launch, and the customers will be no wiser about the missing 20 percent. Structuring a project in this way requires skillful planning up front.

During the prank, plausible deniability is important. Act like it is real, or act like you don't see it, or act like you weren't involved. Do, however, include a link to a page that explains that this is just a joke. They say a joke isn't funny if you have to explain it; if someone doesn't realize it is a joke, that can lead to unfunny situations and hurt feelings. This is the Internet, not Mensa.

Perform a project retrospective.⁵ After the prank, sit down with everyone involved and reflect on what went well, what didn't go well, what should be done the same way next time, and what should have been done differently. Publish this throughout the organization. It not only makes everyone feel included, but it also educates people about how to do better next time. Yes, you may have overloaded the network and created an outage, but if everyone in the organization learned from this experience, your organization is now smarter. Every outage that results in organizational learning is a blessing. If you hide information, the organization stays ignorant.

CASE STUDY: THE MUSTACHE PRANK

One of the most successful AFPs I was involved with was at a previous employer. Managers had been on a teleconference for an hour brainstorming ideas for an AFP.

They wanted one that would be visible only to employees. There's nothing less funny than managers trying to write a joke, so they turned to me. I was a half-manager so they assumed I'd have a half-funny suggestion.

After listening to the ideas they had so far, I was not impressed. They were irrelevant, not topical; silly, not absurdist. Obviously, they did not have the benefit of reading this article.

I thought for a moment. What was the most recently controversy? Well, facial-recognition software was becoming good enough and computationally inexpensive enough that it was making the news and starting a lot of ethical debates.

I blurted out, "Hey, didn't we just purchase a company that makes facial-recognition software? You'd think a smart bunch of people like that would be able to accurately place mustaches on all the photos in the corporate directory."

There was a short pause in the conversation. Then one manager said, "We just moved those people into my building. They sit down the hall from me." Another manager chimed in that he manages the team that runs the corporate directory. Another manages the operations people for it. Another manages the helpdesk most likely to receive any complaints.

Soon, we had a plan.

We started meeting weekly. We wrote a design doc that spelled out how the AFP would work, how we would shut it off after 24 hours, and, most importantly, how individual people could opt out if they complained. A project manager was assigned to coordinate people on three different

continents to make it all happen as expected. HR and executive management signed off on the project.

This was long before social media apps were doing this kind of thing, so the primary question we kept getting was, “Is this really possible?”

Was it technically feasible? Yes. It turns out the free software development kit that the company provided included a mustache-placement API. “Mustaching a person” was the demo they used to sell the company.

By the time April 1 rolled around, a new set of photos was prepared and ready to be swapped in. The helpdesk was trained on how to revert individual photos.

The prank was a huge success. Everyone thought it was hilarious, except for one person who complained and opted out.

Afterwards, we wrote up a retrospective and thanked everyone involved. In such a highly distributed company, this was the best way to let everyone involved “take a bow.”

LAUNCH IT LIKE IT'S HOT

If an AFP will have significant resource needs, load testing is important. Everyone knows how to do load testing: simulate thousands of HTTP requests and take measurements. Find and fix the bottlenecks and repeat until you are satisfied.

You also need to plan for the situation where the AFP goes viral and receives 10 times or 100 times more users than you could ever expect. The easy strategy here is simply to plan on disabling the AFP, but it would be disappointing that the reward for success was to turn the feature off.

Facebook uses a technique called a “dark launch:” testing a feature by first launching it invisibly.

Fixing such a situation is difficult because normal solutions might take weeks to implement and April Fools’ Day lasts only one day. If you fix a problem and relaunch the next day, you’ve missed the boat.

Facebook is in a similar situation when launching real features because there is a lot of press around a new feature and Facebook needs to get it right on the first try. When Facebook was new, growth was slow and bottlenecks could be fixed by simply fixing them at the pace Facebook was growing. By 2008 Facebook had millions of users, and a new feature would go from 0 to millions of users within hours. There would be no time to fix unexpected bottlenecks. A failed launch is highly visible and embarrassing, often becoming front-page news. There is no way, however, to build an isolated system big enough to perform load testing.

To solve this problem, Facebook uses a technique called a “dark launch”: testing a feature by first launching it invisibly. For example, Facebook launched Chat six months early but made it invisible (CSS display: hidden). The HTML and JavaScript code was in your browser, but it did not display itself. A certain percentage of users received a signal to send simulated chat messages through the system. The percentage was turned up over time so that developers could spot and fix any performance issues. By the time the feature was made visible (and the test messages were disabled), Facebook’s engineers were confident that the launch would not have performance problems. It is suggested that nearly every feature that Facebook will launch in the next six months is already running in your browser.⁴

Google did something similar before launching IPv6 connectivity; your browser was running invisible JavaScript that tested whether your ISP connection would fail if IPv6 was enabled. Worries were for naught, but the test increased confidence before launch.

Stack Overflow dark launches new ad-serving infrastructure. When launching major features, we first use the system to transmit house ads that are invisible to users. Once performance is verified, we make the advertisements visible. Sadly, we didn't use this technique when launching StackEgg, but now we know better.

PRANKS WITH MINIMAL OPERATIONAL IMPACT

Technical issues can be avoided with proper testing, but there is a strategy that avoids the issue altogether. Simply create a prank that has no operational impact, or directs the impact elsewhere.

The “Dance Dance Authentication” example is one such prank. The prank was simply a blog post and a link to a YouTube video (<https://www.youtube.com/watch?v=VgC4b9K-gYU>). This doesn't entirely avoid the issue, but if your success ends up overloading YouTube's network, at least it isn't your problem.

You can also simply take an existing feature and create an alternative explanation or history for it. For example, you may have heard of “the teddy bear effect.” Many have observed that often the act of asking a question forces you to think out enough details to realize the answer yourself. In Bell Labs folklore there was a researcher known for helping people with research roadblocks. People would come to him for suggestions. By listening, they would

come up with the answer themselves. Once, he left on a long vacation and left a teddy bear on his desk with a note that read, “Explain your problem to the bear.” Many people found it was equally effective. (Lately, the Internet has started calling this “the rubber duckie effect.”)

Suppose you run a question-and-answer website: some users post questions, and other people post answers. Suppose also that the website has a feature that permits people to write up the answers to their own questions. A very simple but effective AFP would be to rename this feature “teddy bear mode” and write a blog post claiming this to be an entirely new feature, based on the power of a teddy bear’s ability to help solve technical issues.

Related articles

➡ The Burning Bag of Dung - and Other Environmental Antipatterns
And you think you have problems?

Phillip Laplante, Penn State University
<http://queue.acm.org/detail.cfm?id=1035617>

➡ 10 Optimizations on Linear Search
The operations side of the story

Thomas A. Limoncelli
<http://queue.acm.org/detail.cfm?id=2984631>

➡ Ray tracing Jell-O brand gelatin

Paul S. Heckbert, Pixar
<http://dl.acm.org/citation.cfm?id=42375>

SUMMARY

Successful AFPs require care and planning. Write a design proposal and a project plan. Involve operations early. If this is a technical change to your website, perform load testing, preferably including a “dark launch” or hidden launch test. Hide the prank behind a feature flag rather than requiring a new software release. Perform a retrospective and publish the results widely.

Remember that some of the best AFPs require little or no technical changes at all. For example, one could simply summarize the best practices

for launching any new feature but write it under the guise of how to launch an April Fools' prank. That would be hilarious.

References

1. Dumke-von der Ehe, B. 2015. The making of StackEgg; <http://balpha.de/2015/04/the-making-of-stackegg/>.
2. Kottasova, I. 2016. Google's April Fools' prank backfires big time. CNNtech; <http://money.cnn.com/2016/04/01/technology/google-april-fool-prank-backfires/index.html>.
3. Pike, K. 2017. Stack Overflow unveils the next steps in computer security. Stack Overflow Blog; <https://stackoverflow.blog/2017/03/30/stack-overflow-unveils-next-steps-computer-security/>.
4. Rossi, C. 2011. Pushing millions of lines of code five days a week. Facebook; <https://www.facebook.com/video/video.php?v=10100259101684977>.
5. Stack Exchange Network Status. 2015. Outage postmortem: March 31, 2015; <http://stackstatus.net/post/115305251014/outage-postmortem-march-31-2015>.

Thomas A. Limoncelli is the site reliability engineering manager at Stack Overflow Inc. in New York City. His books include *The Practice of Cloud Administration* (<http://the-cloud-book.com>), *The Practice of System and Network Administration* (<http://the-sysadmin-book.com>), and *Time Management for System Administrators*. He blogs at *EverythingSysadmin.com* and tweets at @YesThatTom. He holds a B.A. in computer science from Drew University.

Copyright © 2017 held by owner/author. Publication rights licensed to ACM.