Let’s Find Someone to Blame

ANDY SEELY

Andy Seely is the manager of an IT engineering division, customer-site Chief Engineer, and a computer science instructor for the University of Maryland University College. His wife Heather is his init process and his sons Marek and Ivo are always on the run queue.

andy@yankeetown.com

If two people are in a boat and lost at sea, both have to row to survive; one doesn’t get to be the captain and the other the sailor. In a large organization with layers of management and silos of responsibility, figuring out individual responsibilities and root cause of a failure is a lot more difficult than just blaming someone and moving on. Assigning blame is actually pretty easy. Truly understanding failure and finding a way to appease the top while improving the organization’s overall effectiveness takes real management skill.

The Manager’s Problem: The Product Release Failed

The VP of communications and technology came to me after we got news about a problem with our latest product release. “I want you to deal with your engineer. He totally messed this up.” That’s my job. The engineer works for me. I’m responsible for his actions. The engineer didn’t do anything wrong, but the VP needed blood from his own organization and had already decided who he was going to blame: my engineer.

I’m a senior manager in the organization, but it’s not like I own the company. At the executive level above me, vice presidents like mine can have motivations that are sometimes mysterious. My goals are simple: empower people to do their best work towards making the systems perform business functions correctly and within performance parameters, as cheaply as possible. Sometimes that means knowing people, understanding their motivations, and clearing a path for them. And sometimes that means taking a bullet. This day, I took the bullet.

The Manager’s Choice: Assigning Blame or Understanding the Bigger Picture

It’s a complex organization, with multiple echelons of the organization and several management chains involved. At least six people are involved in the product, not including their individual managers and chains of command. It’s a complex product. From requirement, to build, test, security, QA, release, to deployment, through the occasional post-production error, the product passes through a lot of gates and a lot of hands. The process is mature and usually works well, but when something goes awry it’s difficult to find clear fault. In my opinion, quick blame is really a luxury anyway; understanding the true root cause of a failure is always about learning and improving and, honestly, sometimes about blame.

My investigation showed that our release was done correctly, by the book, with no problems. Maybe it was a little rushed because of a shorter than usual operations deadline. Maybe the engineer who puts the package together had this “normal rush job” to do and he had another rush job to do at the same time. Maybe the security review was done by the second-string technician this time. Maybe there were externalities for which we didn’t test, because our test environment is not set up for everything under the sun. Maybe a lot of things happened, but ultimately the release passed all the gates and met the deadline.

We only got word of a problem after the release was in production for a few days. A user of a remote service provided by an external company was getting failures that were traced back to a local Java dependency. We had updated our Java Runtime Environment to the current
release that is considered a failure by those above me and a success by those below me. I am dealing with a product that I'm the manager of the division. I work for a VP and I have branch managers who work for me. I am dealing with a product release that is considered a failure by those above me and a success by those below me.

There are three ways this might play out.

1. I defend my team to my VP. They did it right, the release was textbook-correct, and the failure case is not on them, it’s on the external organization that didn’t update their code to work with ours. The result will be that the VP will have to show his bosses he took action, and since he can’t control the external organization he’ll look at me as the thing he can control. After all, if the release was both “textbook” and “failure,” then the textbook, my my textbook, was flawed and I refused to take action. I sacrifice either my credibility or my job, and neither result will actually help improve future releases.

2. I act as pass-through for the heat. I write up the integration branch manager, and then he writes up his engineers who built the release. Or I just save him the trouble and write him up and fire his guy for him. Or I fire him and his guy. Make an example out of everyone and prove myself to be “he who manages by fear,” creating a demoralized workforce. This path is easy. Holding others accountable rarely costs you your own skin, as long as you’re willing to blame employees for causing their own demoralization. This approach will result in losing the people whom you have depended upon the most, who know the systems and processes the best, and it will set the stage for a working environment where the only people who stay are those for whom fear is actually an effective motivation.

3. I find the middle ground where my VP can get satisfaction, my crew can be proud of their work, and people can get the chance to improve the process and release failure-resistant products. As a side effect, I can show organizational maturity through flexibility and introspection. I stand in front of the VP and take the blame, which is not entirely misplaced because it’s my organization’s release that was found wanting, but I defy the demand for counseling or firing people. I turn to my team and hold them accountable, without blaming them. I challenge them to find a way to prevent this type of failure in the future. Not just this specific failure, but to define this as a class of failure and fix the process. Evolve.

**Empower and Challenge People to Get Results that Really Matter**

If done well, the middle-ground approach sets the stage for future success and helps to mature the organization. The team sees that the manager took responsibility and didn’t just pass blame directly through. A team that already takes pride in their product will respond well to a challenge to make that product stronger.

I’m the manager. I take the heat and hold the line, while giving the team a new goal. The team sees how much I have on the line, and they’ll work hard for me because I put myself out there for them, and they’ll produce an improved product. My VP gets what he needs to answer his own masters, and maybe is slower to seek blood the next time. And me? I get the pleasure of having that rare opportunity not to be the boss, not to be a manager, but to be a leader, equal parts showing the way and being fully invested in the outcome of the whole team. As a leader, if my own skin isn’t in the game then nothing I do really counts.

**How Did It End?**

We found the actual culprit for the failed release. As everyone had said, it was a textbook product release. We discovered that a chapter of our textbook was, indeed, flawed. Every application had passed its respective application validation checklist, but there was no governance for how those checklists were reviewed and updated. We had been relying on sysadmins and engineers to create the validation checklists, and in some cases they had no idea how the applications were actually used. This added step gave us some immediate benefits. We gained improved user engagement, which gave us greater understanding of how our product was used, and every release was now checked by the people who actually depend on it. Now if a user discovers a post-release product failure in the future, he’s vastly more likely to call his new friends in the engineering shop than to call the CEO. Through understanding real responsibility and holding the line on the blame game, we found the right path to keep our best people on the team and deliver a better product. I’m the manager. That’s my job.
The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks. The Symposium will include a 3-day technical program with refereed papers, invited talks, posters, panel discussions, and Birds-of-a-Feather sessions.

The following co-located events will precede the Symposium.

EVT/WOTE ’14: 2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections
August 18–19, 2014
www.usenix.org/evtwote14

USENIX Journal of Election Technology and Systems (JETS)
Published in conjunction with EVT/WOTE
www.usenix.org/jets

CSET ’14: 7th Workshop on Cyber Security Experimentation and Test
August 18, 2014
www.usenix.org/cset14
Submissions due: April 25, 2014

NEW! 3GSE ’14: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education
August 18, 2014
www.usenix.org/3gse14
Invited submissions due: May 6, 2014

FOCI ’14: 4th USENIX Workshop on Free and Open Communications on the Internet
August 18, 2014
www.usenix.org/foci14
Submissions due: May 13, 2014

HotSec ’14: 2014 USENIX Summit on Hot Topics in Security
August 19, 2014
www.usenix.org/hotsec14

HealthTech ’14: 2014 USENIX Summit on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies
August 19, 2014
www.usenix.org/healthtech14
Submissions due: May 9, 2014

WOOT ’14: 8th USENIX Workshop on Offensive Technologies
August 19, 2014
www.usenix.org/wot14